Automated Verification of Object Petri Nets based on Transformation, Unfoldings and SAT Solving

Abdullahi Ismaila Jihad

A submission presented in partial fulfilment of the requirements of the University of South Wales / Prifysgol De Cymru
for the degree of Doctor of Philosophy

School of Mathematics and Computing

May 2018

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank Allah the Beneficent the Merciful for the life, health and sustenance.

I would like to thank my supervisor Dr. Bertie Müller for the following reasons: For his help, for his patience and for his suggestions on all aspect of academic life, for his great efforts in providing good guidance and research training and was enthusiastic about my work.

Special thanks are due to the person who has been involved in my PhD pursuit as second supervisor: Prof. Janusz Kulon.

I am thankful to the university administrative staff members, the resource office staff members, and technicians.

These acknowledgements would be incomplete without thanking Dr. Elaine Huntley, Llinos Spargo, Carol and the entire staff members of the Graduate Research Office: Thank you!

Finally, I am indebted to the National Information Technology Development Agency (NITDA) that has given me the choice and chance with full funding to pursue what I desired. My late parents, who have built my educational foundation; my wife, Hassana Mamuda, who has always been at my side in all the ups and downs, without her support the dissertation would have been incomplete. My daughters Amina and Khadija and my son Muhammad Kabir for their patience. My brothers and sisters were encouraging towards my studies. My friends Abdulkareem Karasuwa, Muntari Abdullahi (Baba Atiku), Uztaz Yunsa Haruna, Muuhammadu Sirajo, Yushe'u Bagobiri, and Habibu Sani Muhammad have always been very helpful in giving me confidence.

Once again I thank you All! Thank you a lot!

Abstract

Object Petri nets are a Petri net framework that follows the nets-within-nets paradigm introduced by Valk. In Object Petri nets the token of ordinary Petri nets are allowed to be Petri nets themselves. This formalism and similar formalisms that allow to interpret the tokens of an ordinary Petri net as a Petri net are suitable for modelling application systems where the mobility and interactions between cooperating agents are of importance. In large information system, agents are software components that are capable of performing autonomous actions in some environment in order to satisfy their design objectives. To this end, several agents communicate and interact in order to solve a complex problem.

With Object Petri nets it is possible to model the mobility and dynamic interactions among agents based on their hierarchical structures: at the lower level each agent and its internal behaviour is represented by an ordinary Petri net called an *object net*, and agents can move and interact within their environment at the higher level which is also represented by an ordinary Petri net called the *system net*. Each place in the system net represents a location where agents can reside and mobility of agents is modelled by the movement of the object nets from one place to another. Interactions between the system net and object nets are possible via simultaneous firing of transitions.

Generally, ordinary Petri nets offer a clear way of specifying complex systems, while at the same time retaining an intuitive and compact graphical representation. This allows for the use of well-established analysis techniques to verify their desirable properties. In contrast, object Petri nets, however, are beneficial to modelling but lack generally adopted verification techniques. The main challenge in this regard is posed by the highly expressive nature of of the underlying class of Petri nets utilised in the object Petri net. Some modifications have to be enforced in order to make verification feasible. This thesis aims at making a contribution by establishing a path to formal verification of properties of a slightly modified version of the Object Petri nets.

Consequently, the main goal of this thesis has been the development of a technique to systematically define the foundations for Object Petri net transformation into 1-safe

equivalent ordinary Petri net in such a way that employing only affordable resources can handle important questions regarding the verification of various properties using model checking. This goal has successfully been reached in terms of the following contributions summarised below:

- Definition of a modified version of a class of Object Petri nets called elementary reference-net system, as formal representation of RENEW nets than previously studied formalisms of the nets in the RENEW tool.

- Theoretical results for transforming this class of Object Petri nets into a single low-level Petri nets in such a way that application of existing techniques and tools known for low-level Petri nets can handle important questions regarding the verification of various properties using model checking (establishment of a set of transformation rules, and the computational complexity analysis for the transformation).

- Development of a software tool (ERStoPTnet) that permits the automatic execution of the above mentioned set of transformation rules described in the formalism of *Petri Net Markup Language* PNML to obtain an equivalent 1-safe Petri net.

- Development of a generic software tool (PrefixtoCNF) that allows the encoding of prefixes generated by a Petri net unfolder as a propositional satisfiability formula that can be given to a SAT solver for the purpose of model checking the low-level Petri nets in particular of the transformed nets and any bounded low-level Petri net in general, specifically for reachability and deadlock problems.

- Experimental results obtained by implementing ERStoPTnet and the PrefixtoCNF.

- A comparative analysis using results from our technique with an existing established technique proposed for model checking Petri net based models using prefixes of unfolding which shows the feasibility of our approach.

Chapter 1.

Introduction

Object Petri nets are a formalism introducing the concept of nesting to Petri nets. This allows mobility of objects, and by design of the transition occurrence rule, permits interaction between these objects. Despite the fact that object Petri nets are highly expressive in their general form, they are suitable to model applications in, for example, an *agent* context. In large information system, agents are software components that are capable of conducting autonomous actions in some environment in order to satisfy their design objectives. An agent will typically sense its environment and have a repertoire of actions that can be executed to modify the environment, which may appear to respond non-deterministically to the execution of these actions (Nilsson (1998); Weiss (1999a); Woolridge (2001)). An intelligent agent or autonomous agent is an entity that presents some degree of autonomous flexibility by being reactive, proactive and some times sociable (Weiss, 1999b). Meanwhile, the human society is becoming increasingly dependent on the application of agent systems in safety-critical, hazadous, or high impact domains (e.g, mission control, air-traffic control, autonomous satellite, nuclear reactor, and financial industry). Therefore, the correct behaviour and reliability of hardware and software used to specify agent systems is often of paramount importance. This is due to the fact that a failure of such a safety-critical application can necessitate major casualty; even when human lives are not involved, implementation mistakes can sometimes lead to substantial economic loss. This call for appropriate verification tools to provide adequate development of these software/hardware agents.

In modern multi-agent systems several agents can communicate and interact in order to solve a complex problem. This is evident for example, in the Internet and its numerous useful agent applications, which for many people have become part of their daily lives. The complexity of such a system arises from the fact that not only do we expect agents in them to make decisions in situations that are not anticipated aforehand or prior to

execution, agents also interacts with other complex agents (possibly with humans), interact with organisations, and are located in a non-deterministic, inaccessible, dynamic and continuous environment. In general, an agent system is expected to work correctly in a dynamic large-scale, complex environment by having autonomy, adaptability, robustness, flexibility and mobility. Mobility means that an agent be able to move from one location to another. For instance, a mobile device may have to move from one location to another while the owner that uses the device, moves. Another example is a scenario where a code snippet might traverses a firewall in a computer system. Accordingly, mobility and movement is not only meant physically. An important point of mobility is that there can be many locations and agents can move from one location to another one. In many formalisms, for example, the one employed in this work and the formalism of Fagin et al. (2004), there exist a unique environment, the one of the higher level at the hierarchy, which contains all other agents and environments, thus allowing for a uniform description of a system of agents. It is relevant to envisage that mobility in agent systems will be of paramount importance in the mere future. However, the large number of potential interaction with humans or with other computational devices as well as their possible behaviours can make them difficult to understand and difficult to implement. Consequently, suitable *modelling languages* and *verification techniques* that allows developers to check desirable properties of the model are two main factors worth considering for designing such systems correctly. A natural background to model agents is to view them as a set of autonomous mobile entities working concurrently. In this viewpoint, models able to handle concurrency and mobility seem a natural optimal option.

Several modelling modelling formalisms for describing and studying concurrent reactive agents exit. Most frequently used formalisms are process calculi (Milner (1999)) and Petri nets (Petri, 1962). The former provide a framework with a variety of algebraic laws which can be used for the specification of interactions, communications and synchronizations between a collections of independent processes. The latter are a graphical and mathematical modelling formalism for describing and studying information processing systems characterized as being concurrent, asynchronous, distributed and non-deterministic. As a graphical tool, Petri nets provides visual communication aid, and as a mathematical tool it is possible to set up state and, algebraic equations respectively. Most prominent of process calculi dealing with concepts of nesting and mobility are -calculus by (Nilsson, 1998), the Ambient calculus by Cardelli and Gordon (1998), and the Seal Calculus by Castagna et al. (2005) among others.

An advantage of Petri nets is that they allow basic aspects of concurrent systems to be identified both conceptually and mathematically, while retaining intuitive semantics of execution. In all of the process calculi, the states of the system are fundamental notions

while global states are not fundamental notions, as they can be derived from their local counterparts. Petri nets are one of relatively few formalisms admitting the true concurrency semantics, i.e., they can model concurrent execution of several actions directly, in contrast to the interleaving semantics of concurrency, where such an execution is represented by a set of sequential runs, each obtained by successively permuting independent and adjacent actions.

To deal with modelling of large and complex systems, ordinary Petri nets called *Place/-Transition nets* (P/T-nets for short) are extended in various directions: Several concepts of Petri nets with time have been proposed by assigning firing times to the transitions and/or places of Petri nets Molloy (1982). Simalar proposals can also be found in Holliday and Vernon (1987); Merlin and Farber (1976). Coloured Petri nets (CPNs) were introduced by Jensen (1981, 2013) as an extension to P/T-nets which preserves the basic phenomenon of Petri nets and allow tokens of different data types called colour sets. This often allows for more natural and high-level models than using P/T-nets. More recently, designers have made further efforts towards higher expressiveness that has led to the development of the so-called *Nets-within-nets* paradigms first introduced by Valk (1998) and reviewed in Valk (2003). This formalism allows various structured objects as tokens, called object nets, including P/T-nets, CPNs or even subsidiary OPNs (which themselves can have OPNs as tokens, and consequently leads to a nested system of nets). Certain formalisms of this kind are the subject of this thesis. Nets-within-nets have been rigorously investigated in the Petri net literature and are proven to be suitable formalism for modelling the structure and behaviour of active and reactive systems including their mobility, interaction and distribution (see e.g.,Cabac (2010); Köhler and Farwer (2007); Köhler and Rölke (2004); Valk (2003)).

With nets-within-nets, it is possible to model the mobility and dynamic interactions among agents based on their hierarchical structures: at the lower level each agent and its internal behaviour is represented by a P/T-net called an *object net*, and agents can move and interact within their environment at the higher level which is also represented by a P/T-net called the *system net*. Each place in the system net represents a location where agents can reside and mobility of agents is modelled by the movement of the object nets from one place to another. Interactions between the system-net and object nets are possible via simultaneous firing of transitions labelled with corresponding channels (Synchronized transitions). Transitions that are not labelled with channels can fire autonomously or concurrently to other enabled transitions (see (Köhler and Rölke, 2004) for example). Without this viewpoint of nets as tokens, the modeller would have to encode the agent differently, e.g. as a data-type. This has the disadvantage that the inner actions cannot be modelled directly, so, they have to be lifted to the system net, which seems

quite unnatural. By using nets-within-nets it is possible to investigate the concurrency of the system and of the mobility of agents in a model without losing the abstraction needed.

The merits of the application of formal methods to model systems that are characterised as being concurrent in general, and systems that capture mobility in particular, is the possibility to use formal verification techniques to analyse them. Two well-known approaches to formal verification are: theorem proving (Goubault-Larrecq and Mackie, 2001), and model checking Clarke et al. (1999). Theorem proving, despite the fact that it is very general, is semi-automatic, demands for extensive human interaction. Also, theorem proving requires high knowledge in logics, which is the central limitation to the widespread use of this approach in the industry.

In contrast, model checking which is purely automatic, applies a computer to explicitly generate and explore each possible state that a system under verification could reach, and then checks compliance with the specification express in temporal logic, for each of these states. If the exploration of the state space terminates and no state that violates the property specification under consideration is encountered model checking has established that the system is correct with respect to the specification. If a state is encountered that violates the property under consideration, an execution path that leads from the initial state to the state that violates the specification is presented to the user. The user can then obtain useful debugging information, and adapt the model accordingly.

Historically, the implementation of model checking used explicit representation of these many possible ways in which a system would behave, using the *state transition graphs* and tries to search through these graphs with efficient graph traversal techniques. This approach subject model checking to the so-called state space explosion (SSE) problem, meaning that the number of states needed to represent the system accurately may easily exceed the amount of available computer memory, i.e. even a very small specification of a design contain a large number of execution paths. SSE problem, is a major drawback and it is one of the main bottle-neck hindering the vast adoption of model checking in practice.

As a panacea to the SSE problem, techniques that used *symbolic* state space exploration came into being. They can roughly be classified as aiming at an implicit compact representation of the full state space of a system. (see Bryant (1986); Raimondi and Lomuscio (2004)), or as an explicit generation of its reduced representation (e.g., abstraction (Clarke et al., 1994)) and partial order reduction techniques that uses partial order view of concurrent computation to represents system states implicitly using acyclic net. In symbolic

model checking, a breadth first search of the state space is effected through the use of ordinary binary decision diagram, or BDDs. The prominent among these techniques is the work by Ken McMillan (1992), who in his PhD thesis Symbolic Model checking attacked the State-space explosion problem in two ways. Firstly, he proposes to use BDD as data structure for implicitly compact representation of the set of global states of a reactive concurrent system. Secondly, McMillan showed how to algorithmically construct a finite initial part of the unfolding (called finite complete prefix of unfoldings) that preserved as much information as the unfolding itself and he applied it to check deadlock freedom and conformance.

Interstingly, since the beginning of the 21st century, the algorithms for constructing prefixes of unfolding have been greatly improved as described in (Khomenko and Koutny (2001); Esparza et al. (2002); Esparza and Heljanko (2008)). These improvements have paved the way for a new type of model checking technique, *bounded model checking* with satisfiability solving (James and Roggenbach, 2011), and (Esparza and Heljanko, 2008). This technique can be applied to check for both safety and liveness properties, where the verification of a safety property involves checking whether a given set of states is reachable, and the verification of a liveness property involves detecting loops in a systems state transition graph. Constructive experimental results with bounded model checking have been obtained for safety properties. One simple, yet very important type of safety property is an invariant, a property that must hold in all reachable states. Noticeably, if a sequence of states can be found that begins at an initial state and ends in a state where the supposed invariant is false, it suffice to know that the property is not an invariant. It turns out that such searches for counterexamples can be done with remarkable efficiency with bounded model checking on designs that would be difficult for BDD based model checking. The strength and the capacity increase of bounded model checking has made its applicability attractive in the industry. Success of these advantages is the fact that satisfiability solvers, such as SATO Zhang (1997), MinSAT Ansótegui et al. (2012) and MaxSAT Ansótegui et al. (2009) , rarely require exponential space, while BDDs frequently do.

However, despite the success in using nets-within-nets to model systems which are rather complicated to model with ordinary Petri nets, most behavioural properties which are decidable for ordinary Petri nets models, become undecidable for nets-within-nets models. For instance in Köhler and Rölke (2004) it was proven that reachability and liveness become undecidable problems for a class of object nets which restrict the nesting of nets to the depth of two, while in Lomazova and Schnoebelen (1999), it was shown that reachability are undecidable properties for nested nets. Consequently, this is a severe drawback if developers needs to perform some kind of analysis which requires similar

properties.

The development described in this thesis is based on the *Elementary Object Nets System* (EOS), particularly, safe elementary object net system (safe-EOS) which was introduced in Heitmann and Köhler-Bußmeier (2012) as a nets-within-nets framework for modelling systems that capture both *nesting*, *mobility* and *interaction* of agents. In safe-EOS, the nesting of nets involved in the model is restricted to depth of two levels, i.e. there is a Petri net, called the *system net*, whose tokens are allowed to be P/T nets called *object nets*. Object nets are tokens with internal structure and inner activity and they can move from one place of the system net to another. Interactions between system net and object nets are performed by *synchronisation* of transitions. Transitions in the system net and object nets that must fire synchronously are labelled by corresponding channels. Transitions without a label can fire autonomously.

In the safe-EOS model the set of all reachable states is bounded, therefore no decidability issues exist since it is always possible to compute the reachability graph. For example, in Heitmann (2013) it was proven that not only reachability but every property that can be expressed in the temporal logics LTL or CTL can be analysed in polynomial space, generalising a result in Esparza (1998) for 1-safe Petri nets (1-safe nets can be seen a particular case of the definition of Petri nets where all arcs have a weight of 1, and there is at most one token in each place in every reachable marking). This phenomenon is sufficient to affirm that all the techniques available for the analysis of 1-safe Petri nets are also applicable to the safe-EOS. On the other hand, the main drawback of safe-EOS models is the existence of some constraints in the definition that limit their expressiveness for *automatic verification* purposes. For instance, there are no arc inscriptions associated to arcs of a safe-EOS models making them impossible to establish exactly which net tokens to remove from the input places of a system net transition and which ones to add to its output places when it fires. Also the labelling of the transitions has to adhere to some restrictions in order to deduce the set of synchronised events. As a result, this formalism requires modification in order to make automatic verification of its properties feasible and align the formalism with the nets used in the RENEW tool.

The question of how to modify the definition of safe-EOS to attain a formalism that still captures the notions of nesting, mobility, and interaction, and that is still expressive in terms of modelling power, while on the other hand it is possible to apply existing automatic verification techniques known for 1-safe Petri nets, particularly the model checking approach, need to be addressed. Consequently, we provide a path to verification of properties of a slightly modified version of the safe-EOS, which we called *elementary reference-net systems* (*ERS*) with reference semantics that is practically relevant and

overcomes fundamental decidability issues with other formalists as shown in Köhler and Rölke (2004) and Lomazova and Schnoebelen (1999).

Compared to safe-EOS, two main additions are introduced for ERS: Firstly, we provide each marked object net located in places of the system net with a *unique name* so that object nets with the same marking can be distinguished. Similar approaches have been taken before, e.g,. (Rosa-Velardo and De Frutos-Escrig, 2007) extend P/T nets with pure name creation and name management. Secondly, we use *variables* from a finite set to label arcs of the system net. When firing transitions, variables are bound to object nets names instead of statically typing system net places. This allows dynamic use of net-tokens without fixing types for places of the system net. An arc variable, is interpreted either as reference to a marked object net (net token) or the black-token net $N_\bullet$, which has an empty structure. As in similar formalisms, we have to distinguish autonomous and synchronous transitions.

For ERS, object net *markings* are not allowed to be empty, and every transition in the system net and the object net must have at least one input place. The same reference can be used as a token in more than one place, meaning that reference semantics assumes a global name space for object nets.

We define some structural restrictions to ensure that new object nets can neither be created nor an existing one destroyed when a transition fires in the system net. Again, we define dynamic restrictions by extending the notion of 1-safe P/T nets to ERS to guarantee that the state space is finite and markings are bounded.

Further to the definition of ERS, we established a set of transformation rules from 1-safe ERS into P/T nets and show isomorphism of the state spaces of ERS with its generated P/T net. Subsequently, we analyse the computational complexity for transforming 1-safe ERS into 1-safe P/T net.

The need for application of *bounded model checking with satisfiability solving* via partial order *(unfolding)* approach for dynamic analysis of resulting transformed low-level Petri net has encouraged and driven the development of this new formalism. We have chosen to integrate an existing, external tool for unfolding the resulting net instead of developing this functionality from scratch. This tool is PUNF (Petri Net Unfolder, Khomenko (2016)). It provides the desired finite complete prefix of unfoldings functionality for 1-safe Petri net in an efficient and accessible way. Thus, all the verification tools employing prefixes can be re-used. This contribution presents the technical and conceptual enhancements to Petri Net-Based Models of Agent-Oriented Software Engineering PAOSE.

Finally, all developed algorithms were implemented and collected into software tools aimed to support verification of Petri net-based models of agent-oriented software.

Consequently, the aim of this research is to create a path to automated verification of a class of nets-within-net paradigm suitable for modelling the nesting and mobility of agents systems by transforming such systems into equivalent behavioural low-level Petri nets and subsequently applying formal verification techniques known for low-level Petri nets; in more detail, the objectives of this thesis are:

- **To establish a path to verifying Object Petri Net Systems**: This task comprises modification and definition of a new class of object net system which we called *Elementary Reference Net System*, establishing set of rules for transforming ERS into a behaviourally equivalent 1-safe P/T net in such a way that established tools can handle important questions regarding the verification of behavioural properties.

- **Develop a transformer from ERS into 1-safe P/T net**: This task consists of the development of a software tool for automatic transformation of ERS into a low-level P/T net which we called ERStoPTnet. This tool takes as input a *Petri Net Markup Language* (PNML) representation of ERS and automatically generates two output formats: One of these outputs is a P/T net representation in PNML format that can be exported into a high-level Petri net editor and simulator called RENEW; the second output is a textual representation that the unfolding tool called PUNF requires for generating *complete finite prefix of unfoldings* for the transformed nets.

- **Develop SAT instance builder**: This task involves the development of a generic software tool for creating a *Conjunctive Normal Form* (CNF) version of reachability and deadlock-freeness problems, aimed at using *bounded model checking* technique on the resulting prefixes of unfoldings with a SAT solver.

Correspondingly, the outcomes of the research work presented in this thesis are:

- Definition of a meaningful class of object Petri nets called elementary reference-net system, that represent more closely than previously studied formalisms the nets in the RENEW tool.

- Theoretical results about transforming elementary reference-net system into a low-level Petri net (a set of transformation rules,and the computational analysis result for the transformation, establishment of a relationship between the isomorphic properties of state spaces of 1-safe ERS and 1-safe P/T net). Among such results are the

established Lemmas and prove of a theorem which relate the state space of 1-safe P/T nets and state space of 1-safe ERS.

- ERStoPtnet: a software tool for automatic transformation of ERS into P/T net.

- PrefixtoCNF: A tool that can encode the sequential behaviour of prefixes generated by a Petri net unfolder as a propositional satisfiability formula in Conjuctive Normal Form which will be given to a propositional decision procedure, i.e., SAT solver, to either obtain a satisfying assignment or to prove there is none.

- Experimental results obtained by implementing ERStoPTnet and PrefixtoCNF.

- A comparative analysis with results from our technique and an established technique proposed to combat state space explosion problem in model checking Petri net based models using prefixes of unfolding.

1.1. Structure of this thesis

This work presents theoretical results about modification and redefinition of EOS and transforming it into a low-level Petri net, the development and implementation of transformation tools and model checking the resulting nets, and performing empirical evaluation to show feasibility of our approach. The overall structure of the thesis is depicted in Figure 1.1 on page 11 Specifically:

- Chapter 2 reviews the most important notions, results, and tools relevant for our purpose from set theory, computational complexity theory, and from the theory of Petri net tools.

- Chapter 3 discusses the basic Elementary Object-nets System (EOS). We use this to introduce elementary object systems with an arbitrarily but fixed nesting depth and show that a similar safeness definition as for elementary object systems can be introduced.

- Chapter 4 describes the adaptation of EOS which we call Elementary Reference-net system and the proposed set of transformation rules from ERS into P/T nets showing it is isomorphic to the state spaces of ERS with its generated low-level

P/T net, and subsequently, presents computational complexity analysis results for the transformation.

- Chapter 5 presents how the transformation algorithm was technically implemented and collected into a software tool.

- Chapter 6 describes the translation of reachability and deadlock detection problems to SAT formulae with implementation, and presents an empirical evaluation of the proposed method through comparative analysis of the functional performace of our technique with other techniques for combating state space explosion by using some low-level Petri net benchmarks collected by Corbett (1996).

- Chapter 7 presents open issues and concludes the the thesis.

The topics presented in most of Chapter 2 and Chapter 3 are not novel, but are incorporated as the foundations upon which the results of this thesis are based.

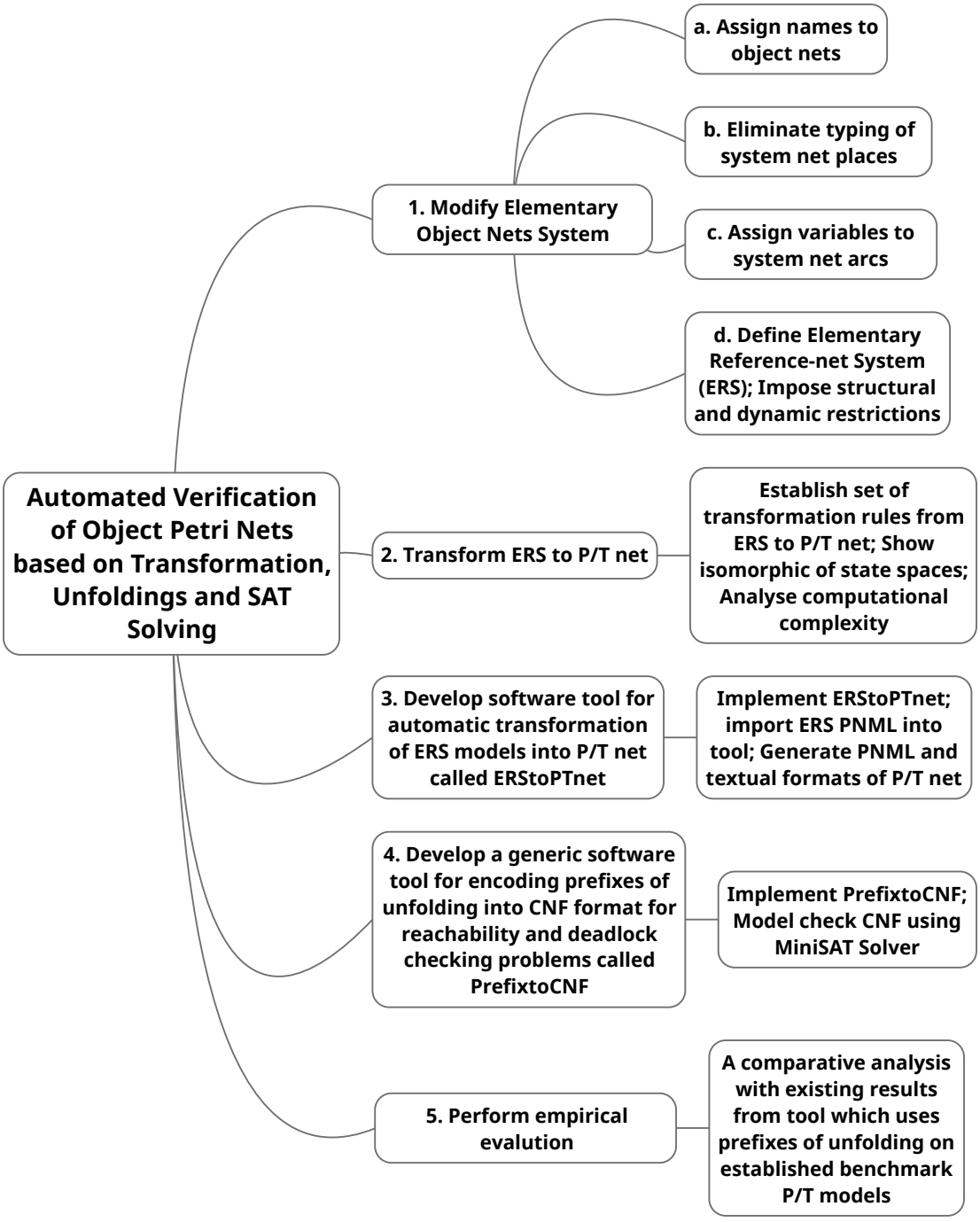

Figure 1.1.: Overall structure of the thesis

Chapter 2.

Fundamentals

In this chapter, we review some of the mathematical preliminaries and most important facts from set theory,computational complexity theory, and from theory of Petri nets, which are relevant for our study.

2.1. Sets, Relations and functions

In this section, we briefly review some of the basic concepts of sets, relations and functions that arise naturally in the analysis of Petri nets. More detail treatments can be found in most books on set theory and discrete mathematics.

2.1.1. Sets

The term set is used to refer to any collection of objects, which are called members or elements of the set. A set is called finite if it contains n elements, for some constant $n \geq 0$, and infinite otherwise. Examples of infinite set include the set of Natural numbers denoted by $\mathbb{N}$, the sets of Integers denoted by $\mathbb{Z}$, the set of Rational numbers denoted by $\mathbb{Q}$ and the set of Real numbers denoted by $\mathbb{R}$.

Informally, an infinite set is called countable if its elements can be listed as first element, second, and so on; otherwise it is called uncountable. For example, the set of integers $\{0, 1, -1, 2, -2, \ldots\}$ is countable, while the set of real number is uncountable.

A finite set is described by listing its elements in expressive way and enclosing this list in braces. If the set is countable, three dot may be used to indicate that not all the elements have been listed. For example, the set of integers between 1 and 100 can be listed as $\{1, 2, 3, \ldots, 100\}$ and the set of natural number numbers can be stated as $\{1, 2, 3, \ldots\}$. A set may also be denoted by specifying some property. For example, the set $\{1, 2, \ldots, 100\}$ can also be denoted by $\{x \mid 1 \leq x \leq 100$ and x is integer $\}$. An uncountable set can only be described in this way. For example, the set of real numbers between 0 and 1 can be expressed as $\{ x \mid x$ is is a real number and$0 \leq$x$\leq 1\}$. The empty set is denoted by $\{\}$ or $\emptyset$.

If A is a finite set, then the cardinality of A, denoted $|A|$, is the number of elements in A. We write $x \in$A if x is a member of A, and $x \notin A$ otherwise. We say that a set B is a subset of set A, denoted by $B \subseteq A$, if each element of B is an element of A. If in addition $B \neq A$, we say that B is a proper subset of A, and we write $B \subset A$. Thus, $\{a, \{2, 3\}\} \subset \{a, \{2, 3\}\}$ but $\{a, \{2, 3\}\} \not\subseteq \{a, \{2\}, \{3\}, b\}$. For any set A, $A \subseteq A$ and $\emptyset \subseteq A$. We observed that if A and B are sets such that $A \subseteq B$ and $B \subseteq A$ then A= B. Thus, to prove that two sets A and B are equal, we need to prove that $A \subseteq B$ and $B \subseteq A$. The union of two sets A and B, denoted by $A \cup B$, is the set $\{x \mid x \in A \, or \, x \in B\}$, The intersection of two sets A and B denoted by $A \cap B$, is the set $\{x \mid x \in A$ and $x \in B\}$.The difference of a set A from a set B, denoted $A \setminus B$, $\{x \mid x \in A$ and $x \notin B\}$. The compliment of a set A, denoted $\overline{A}$, is defined as $U \setminus A$, where U is the universal set containing A, which is usually understood from the context. If A, B and C are sets, then $A \cup (B \cup C) = (A \cup B) \cup C$, and $A \cap (B \cap C) = (A \cap B) \cap C$. We say that two sets A and B are disjoint if $A \cap B = \emptyset$. The Power set, denotes by $\mathbb{P}(A)$, is the set of all subsets of A. Note that $\emptyset \in \mathbb{P}(A)$ and $A \in \mathbb{P}(A)$. If $|A|$=n, then $|\mathbb{P}(A)| = 2^n$.

2.1.2. Relations

An ordered n - tuple $(a_1, a_2, \ldots, a_n)$ is an ordered collection that has a_1 as its first element, a_2 as its second element,$\ldots, a_n$ as its nth. In particular, $2 - tuples$ are called ordered pairs. Let A and B be two sets. The Cartesian product of A and B denoted by $A \times B$, is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. In set representatioon,

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\} \tag{2.1}$$

More generally, the Cartesian product of $A_1.A_2, \ldots, A_n$ is denoted as

$$A_{\times}A_2 \times \ldots A_n = \{a_1, a_2, \ldots, a_n) \mid a_i \in A_i \wedge 1 \leq i \leq n\} \tag{2.2}$$

Let A and B be two non-empty sets, A binary relation, or simply a relation R from A to B is a set of ordered pairs (a, b) where $a \in A$ and $b \in B$, that is $R \subseteq A \times B$, if $A = B$ we say R is a relation on the set A. The *domain* of R sometimes written $Dom(R)$, is the set

$$Dom(R) = \{a \mid \forall b \in B(a, b) \in R\} \tag{2.3}$$

The range of R, sometimes written $Ran(R)$, is the set

$$Ran(R) = \{b \mid \forall a \in A(a, b) \in R\} \tag{2.4}$$

Example 2.1 Let $R_1 = \{(2, 5), (3, 3)\}$, $R_2 = \{(x, y) \mid x, y \text{ are positive integers and } x \leq y\}$ and $R_3 = \{(x, y) \mid \text{ x,y are real numbers and } x^2 + y^2 \leq 1\}$ Then, $Dom(R_2) = \{2, 3\}$ and $Ran(R_1) = \{5, 3\}$, $Dom(R_2) = Ran(R_2)$ is the set of natural numbers, and $Dom(R_3) = Ran(R_3)$ is the set of real numbers in the interval $\{-1 \ldots 1\}$.

Let R be a relation on a set A. R is said to be *reflexive* if $(a, a) \in R$ for all $a \in A$. It is *irreflexive* if $(a, a) \in R$ for all $a \notin R$ for all $a \in A$. It is *symmetric* if $(a, b) \in R$ implies $(b, a) \in R$. It is *asymmetric* if $(a, b) \in R$ implies $(b, a) \notin R$. It is *antisymmetric* if $(a, b) \in R$ and $(b, a) \in R$ implies $a = b$. Finally, R is said to be *transitive* if $(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$. A relation that is reflexive, antisymmetric, and transitive is called *partial ordered*.

2.1.3. Functions

A function f is a (binary) relation such that for every element $x \in Dom(f)$ there is exactly one element $\in Ran(f)$ with $(x, y) \in f$. In this case, one usually writes $f(x) = y$ instead of $(x, y) \in f$ and syas that y is the value or image of f at x.

Let f be a function such that $Dom(f) = A$ and $Ran(f) \subseteq B$ for some non-empty set A and B. We say that f is *one to one* if for no different element x and y in A, $f(x) = f(y)$. That is, f is one to one if $f(x) = f(y)$ implies $x = y$. We say that f is onto B if

$Ran(f) = B$. f is said to be a *bijection* or *one to one correspondence* between A and B if it is both one to one and onto B

2.1.4. Multisets

Definition 2.1 (**Multisets**). *Let $X \neq 0$ be a set. A multiset (or bag) μ is a mapping $\mu : X \longrightarrow \mathbb{N}$, which associates to each, element $x \in X$ a non-negative integer coefficient (or multiplicity) $\mu(x)$. A multiset will be represented as the formal sum of its components: $\mu = \Sigma_{x \in X}\ \mu(x)'x$. We denote by $Bag(X)$ the set of multisets over X. By extending the set operation to multiset we define the addition (+) and difference (-) as follows:*

if μ, μ_1 and μ_2 are multisets defined over the same set X, then:

- $\mu_1 + \mu_2 := \Sigma_{x \in X}\ (\mu_1(x)\ +\ \mu_2(x))'x$
- $\mu_1 \leq \mu_2 \Longleftrightarrow \forall x \in X, \mu_1(x) \leq \mu_2(x)$
- $\mu_1 - \mu_2 := \Sigma_{x \in X}\ (\mu_1(x)\ \text{-}\ \mu_2(x))'x\ \ \mu_2(x) \leq \mu_1(x)$ *and*
- $|\mu| := \Sigma_{x \in X}\ \mu(x)$ *is the cardinality of μ and ϕ denotes the empty multiset (with $\mu = 0$).*

An example of a multiset over $X = \{q, r, s, t\}$ is $\{q, q, r, r, r, t\}_m$ (where the subscript distinguishes set brackets from multiset brackets) or, equivalently, $\mu_1 = 2'q + 3'r + t$. With $\mu_2 = q + 2'r$ we obtain $\mu_1 + \mu_2 = 3'q + 5'r + t$ and $\mu_1 - \mu_2 = 1'q + 1'r + 1't = q + r + t$.

2.1.5. Partially Ordered Sets

A partially ordered set (or poset) consists of a set together with a binary relation that indicates that, for certain pairs of elements in the set, one of the elements precedes the other. Thus, partial orders generalize the more familiar total orders, in which every pair is related.

A partial order is a binary relation $\ $ over a set A which is reflexive, antisymmetric and transitive, i.e.: for all a, b, and $c \in A$:

- $a \leq a$ (reflexivity)
- if $a \leq b$, and $a \leq b$ then $a = b$ (antisymmetry)
- if $a \leq b$, and $b \leq c$ then $a \leq c$ (transitivity)

Let $(A, \preceq)$ be a partially ordered set. We call $(A, \preceq)$ a well-founded set if and only if every non-empty subset of A contains at least one minimal element m with respect to the order $\preceq$.

2.2. Computational Complexity

This section reviews fundamental facts from computational complexity theory which is concerned with classifiying problems based on the amount of time, space or any other resource required to solve a problem. An example of a resource could be the number of processors and communication cost. We consider only the notion of running time of an algorithm, as it is of importance to the time complexity of the algorithm associated with translating ERS into a 1-safe P/T net, and most important computational complexity classes for those problems whose solution output is either *yes* or *no*. A problem of this type is called a *decision* problem which for example, is the *satisfiabilty* problem used by our developed model checking technique. It turns out that we can encode such problems as *languages*. More detailed discussions on the fundamentals of computational complexity theory and their classifications can be in found in Alsuwaiyel (2016), Sipser (2006).

2.2.1. Basic Concepts in Algorithmic Analysis

In this subsection we investigate the mathematical aspects underlying the analysis of algorithms. Specifically, we will describe briefly the analysis of the running time and space required by an algorithm. For more detailed coverage of analysis of algorithms the reader is referred to standard books like the works of (Brassard and Bratley (1988); Manber (1989); Preiss (1999)). For a more wide spread account of algorithms see Lewis and Papadimitriou (1978), and the two Turing Award Lectures of Karp and Rabin (1987) and Tarjan (1987).

Time is indisputably and extremely precious resource to be investigated in the analysis

of algorithms. We will use an example to elaborate this fact. Let us assume that the maximum number of element comparisons performed by a certain sorting Algorithm A, when the input data is a power of 2 is at most $n \log n - n + 1$ and the number of element comparison performed by another sorting Algorithm B, is at most $n(n-1)/2$. For purpose of accuracy, let us assume that each element comparison takes 10^{-6} seconds on some computing device. Suppose we want to sort a small number of input data, say 64. Then the time taken for comparing elements using Algorithm A is at most $10^{-6}(64 \times 6 - (64+1)) = 0.003$ seconds. Applying algorithm B, the time will be $10^{-6}(64 \times 63)/2 = 0.002$ seconds. This, of course, is not perceptible, especially to a beginner programmer whose main concern is to come up with a program that does the job. However, if we consider a larger input data, say $n = 2^{20} = 1,048\,576$ which is typical of many real world problems, we find the following: The time taken for comparing elements using Algorithm A is at most $10^6(2^{20} \times 20 - 2^{20} + 1) = 20$ seconds, whereas, using Algorithm B the time becomes $10^6(2^{20} \times (2^{20} - 1))/2 = 6.4$ days! Without time complexity analysis, one would assume that Algorithm B is faster than Algorithm A when used for sorting the same data elements of input size n. However, this example reveals the fact that time is undoubtedly an extremely precious resource to be investigated in the analysis of algorithms.

2.2.2. Order of growth

The actual time taken by an algorithm to solve a given problem is not always sufficient to say that the algorithm is efficient. This is because there are other factors to consider when analysing the effectiveness of an algorithm. These may include for instance, how and on what device the algorithm is executed, and in what language or programmer's skills or even what compiler is the algorithm implemented in to just to mention a few. Consequently, we should be satisfied with only an approximation of the actual time. However, numerous studies have shown that we do not have to deal with the actual time or even approximate times when evaluating an algorithm's efficiency. This is supported by factors which includes the following: Firstly, when analysing the running time of an algorithm we have to compare its behaviour with another algorithm that solves the same problem. Accordingly, our estimate of time are *relative* as opposed to *absolute*. Secondly, it is required for an algorithm to be machine independent, be capable of being expressed in any language including human languages, and moreover, we should be concerned that our measure of the running time of an algorithm survives technological advancements. Thirdly, our concern is not only on small input data sizes; we are always concerned with the behaviour of an algorithm under study on large input sizes.

In general, when assessing an algorithm, counting the number of operations is more than what is required. As a result of the third factor above, we can move a step further. Because an accurate count of the operations is always very tedious if not impossible, and since we are concerned with the running time for large input data sizes, we might speak about the *order of growth* of the running time. For example, if we can state that, for some constants $c > 0$ such that the running time of an algorithm A when given an input data of size n is at most cn^2, we will see that c becomes insignificant as the size of n increases. The same thought applies to lower order term of functional expressions, as for instance, in the function $f(n) = n^3 \log n + 20n^2 + 5n$. Here, we can see that the larger the value of n the lesser the importance of the contribution of the lower terms $20n^2$ and $5n$. Consequently, we may speak about the running time of algorithm A above to be "in order of " n^2 . Similarly, we say that the function $f(n)$ above is in the order $n^3 \log n$.

Whenever we discard preceding constants and lower order term(s) from a function that represents the running time of an algorithm, we say that we are evaluating the *asymptotic* running time or using the more technical term "time complexity"of an algorithm.

In some cases however, the constant may be important for more detailed analysis especially when comparing the behaviour of one algorithm to another with same running times in the order of $n \log n$ in other to determine which is preferable. In addition, it may be necessary to investigate other factors like the space requirement and input distribution. Using the input distribution is helpful when analysing the behaviour of an algorithm in an average case.

Figure 2.1(a) on page 19 shows some functions that are widely used to express the running time of algorithms. They are called, respectively, *logarithmic, linear, quadratic and cubic.* Higher order, exponential and hyper-exponential functions are not shown in the figure. They are functions that can grow faster than the ones shown in the figure, even for a small size of n.

2.2.3. $\mathcal{O}-$, Ω-, and Θ-Notations

In the followings, we present special mathematical notations widely used to formalise the notions of the order of growth and asymptotic running time of algorithms. These

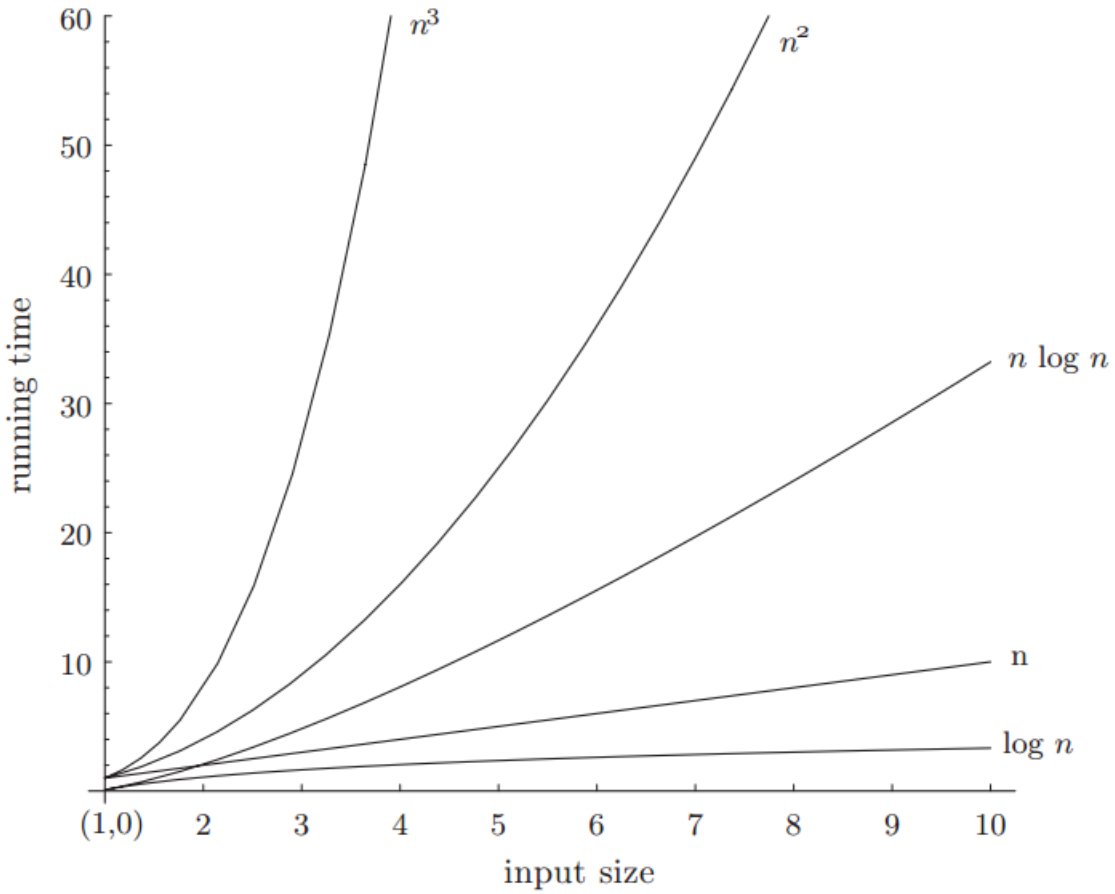

Figure 2.1.: Growth of some typical functions that represent running times (from Alsuwaiyel (2016))

notations provide approximations that make it convenient to evaluate the large-scale differences in algorithm efficiency, while ignoring differences of a constant factor and differences that occur only for small sets of input data.

The idea of the notations, $\mathcal{O}-$notation (read "big-Oh notation"), $\Omega-$notation (read "big-Omega notation") and $\Theta-$notation (read "big-Theta notation") is this. Suppose f and g are real-valued functions of real variables n.

1. If, for sufficiently large values of n, the values of $|f|$ are less than those of a multiple of $|g|$, then f is of order at most g, or $f(n)$ is $\mathcal{O}(g(n))$.

2. If, for sufficiently large values of n, the values of $|f|$ are greater than those of a multiple of $|g|$, then f is of order at least g, or $f(n)$ is $\Omega(g(n))$.

3. If, for sufficiently large values of n, the values of $|f|$ are bounded both above and below by those of multiples of $|g|$, then f is of order g, or $f(n)$ is $\Theta(g(n))$.

Definition 2.2. *Let f and g be real-valued functions defined on the same set of nonnegative real numbers. Then f* ***is of order at least*** *g, written $f(n)$ is $\Omega(g(n))$, if, and only if, there exist a constant $c > 0$ and a natural number n_0 such that $c|g(n)| \leq |f(n)|$ for all real numbers $n \geq n_0$.*

Consequently, if $\lim_{n\to\infty} f(n)/g(n)$ exists, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq 0 \text{ implies } f(n) = \Omega(g(n)).$$

Informally, this definition says that f grows at least as fast as the product of some constant and g. It is obvious from the definition that

$$f(n) \text{ is } \Omega(g(n)) \text{ if and only if } g(n) \text{ is } \mathcal{O}(f(n)).$$

Definition 2.3. *Let f and g be real-valued functions defined on the same set of nonnegative real numbers. Then f* ***is of order at most*** *g, written $f(n)$ is $\mathcal{O}(g(n))$, if, and only if, there exist a constant $c > 0$ and a natural number n_0 such that $c|g(n)| \geq |f(n)|$ for all real numbers $n \geq n_0$.*

Consequently, if $\lim_{n\to\infty} f(n)/g(n)$ *exists, then*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \neq \infty \textit{ implies } f(n) = \mathcal{O}(g(n)).$$

Informally, this definition says that f *grows no faster than the product of some constant and* g*. The* $\mathcal{O}$*-notation is sometimes used in equations as a simplification tool. For example, instead of writing*

$$f(n) = 10n^3 + 14n^2 - 4n + 26,$$

we may write

$$f(n) = 10n^3 + \mathcal{O}(n^2).$$

This is helpful if we are not interested in the details of the lower order terms of the equation.

Definition 2.4. *Let* f *and* g *be real-valued functions defined on the same set of non-negative real numbers. Then* f ***is of order*** g*, written* $f(n)$ *is* $\mathcal{O}(g(n))$*, if, and only if, there exist two positive constants* c_1 *and* c_2 *and a natural number* n_0 *such that* $c_1|g(n)| \leq |f(n)| \leq c_1|g(n)|$ *for all real numbers* $n \geq n_0$*.*

Consequently, if $\lim_{n\to\infty} f(n)/g(n)$ *exists, then*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \textit{ implies } f(n) = \Theta(g(n)),$$

where c *is a constant strictly greater than* 0*.*

An important result of the above definition is that

$$f(n) = \Theta(g(n)) \textit{ if and only if } f(n) = \Omega(g(n)) \textit{ and } f(n) = \mathcal{O}(g(n)).$$

Remarkably, it is important to note that the above notations are not only used to describe the time complexity of an algorithm; they are also used to characterize the asymptotic behaviour of amount of space used by an algorithm. Theoretically, they may be applied in combination with any abstract function. Consequently, we will not attach any measures or meanings with the functions in the complexity analysis presented in section 4.4.2. We will assume in the analysis that $f(n)$ is a function from the set of non-negative integers to the set of non-negative integers.

2.2.4. Complexity Classes

This subsection presents the classification of problems based on the amount of time and space needed to solve a particular problem as opposed to the time and space of a particular algorithm to solve that problem. Thus it summarises the notions to reason about the *complexity of decision problems* and it's based on material from Alsuwaiyel (2016).

A decision problem is a problem which is stated so that its solution has only two outcomes: yes or no. The COLORING problem is an example of a decision problem; given an undirected graph $G = (V, E)$ (where V is a set of vertices and E is a set of edges) and a positive integer $k \geq 1$, COLORING is the problem that questioned whether G is k-colorable?, i.e. can G be colored using at most k colors?

Turing machine, which was orriginally introduced by Alan M.Turing in Turing (1938). offers a universal formalism to reason about the complexity of the algorithms applied in decision problems. A Turing machine is universal in the sense that anything that can ever be computed on a machine can be computed with the Turing machine. In language theory an *alphabet* Σ is a finite set of characters. A language L is simply a subset of the set of all finite length strings of characters chosen from Σ, denoted by Σ^*. Let Σ be an alphabet. for each nonnegative integer n, the set of all strings over Σ that have length n is denoted as Σ^n. Denoted by Σ^+ is the set of all strings over Σ that have length at least 1.

A standard Turing machine operates on a string of symbols presented on a one-way infinite tape which is partitioned into separate cells. Each cell of the tape holds one character from some finite alphabet Σ. The Turing machine has the ability to read and replace the character contained in the cell of the tape currently scanned by it tape-head. The tape-head can read the character in the cell it is positioned at, write a new character to that cell and either move one step to the left, one step to the right or remains on the current cell at each step. The machine then continues in this manner. If during this process the machine reaches a final state the input string of characters is said to be accepted.

Definition 2.5 (**Turing Machine**). *A Turing machine is a* 6*-tuple* $M = (S, \Sigma, \Gamma, \delta, p_0, p_f)$*, where*

- S *is a finite set of states,*
- Γ *is a finite set of tape symbols which includes the special symbol* B *(describing the*

blank symbol) and $\Gamma \cap S = \phi$,

- $\Sigma \subseteq \Gamma \setminus \{B\}$, *the finite set of input symbols,*
- δ *ia a transition function,*
- $p_0 \in S$, *the initial state, and*
- $p_f \subseteq S$, *the final or accepting state.*

In general a Turing machine can be deterministic or non-deterministic. In the case of a deterministic Turing machine M *the transition function is defined as* $\delta : (S \times \Gamma) \rightarrow (\Gamma \times \{L, R, H\} \times S)$ *where the meaning of* $\delta(p, a) = (a', D, p')$ *is that if* M *is in the state* p *reading the symbol* a *from the tape, it replaces* a *with* a'*, moves the tape-head to the direction given by* $D \in \{L, R, H)$ *(move to left symbol on the tape, move to the right symbol on the tape, or hold position) and changes state to* p' *i.e for each tuple from* $(S \times \Gamma)$ *only at most one action is possible.*

In the case of a non-deterministic Turing machine M *the transition function is defined as* $\delta : (S \times \Gamma) \rightarrow 2^{\Gamma \times \{L,R,H\}} \times S$*, i.e if for every* $p \in S$ *and for every* $a_1, a_2, \ldots, a_k \in \Gamma$*, the tuple* $\delta(p, a_1, a_2, ..., a_k)$ *none, or one or more than one action is possible.*

A Language $L \subseteq \Sigma^*$ is decidable by a Turing machine M, if for all input strings $x \in L$, the machine *accepts* x, it is denoted $M(x) = yes$. If for all input strings $x' \in L$ the machine *rejects* the input x' it is denoted $M(x) = no$

For many formalisms there exists an equivalent Turing machines in terms of computational power. For example, if a language L can be accepted by a Turing machine it can also be accepted by another equivalent formalism. The equivalent formalism is also said to be *Turing-complete*.

In order to adequately measure the amount of time used by an algorithm, Turing machines are extended to a multi-tape Turing machines that has k tapes instead of just one, and it has k tape-heads: A k-tape Turing machine, for some $k \geq 1$, is a tuple $M = (S, \Sigma, \Gamma, \delta, p_0, p_f)$ where S, Γ and Σ are as above, and the transition function δ takes into consideration the k tapes of M. Formally, $\delta : (S \times \Gamma)^k \rightarrow ((\Gamma \times \{L, R, H\})^k \times S)$.

Definition 2.6 (Configuration of k-tape Turing machine). *Let* $M = (S, \Sigma, \Gamma, \delta, p_0, p_f))$

be a k-tape Turing machine. A configuration of M is a $(k+1)$-tuple

$$C = (p, w_{11} \uparrow w_{12}, w_{21} \uparrow w_{22}, \ldots, w_{k1} \uparrow w_{k2})$$

, where $p \in S$ and $w_{j1} \uparrow w_{j2}$ is the contents of the j-th tape of M, $1 \leq j \leq k$.

Definition 2.7 (**Computation by a k-tape Turing machine**). *A computation by a Turing machine M on input x is a sequence of configurations $C_1, C_2, \ldots, C_n$, for some $n \geq 1$, where C_1 is the initial configuration, and for all i, $2 \leq i \leq n$, C_i results from C_{i-1} in one move of M. Here, n is referred to as the length of the computation. If C_n is a final configuration, then the computation is called an accepting computation.*

Definition 2.8. *The time required by a Turing machine M on input x, denoted by $T_M(x)$, is defined by:*

- *If there is an accepting computation of M on input x, then $T_M(x)$ is the length of the shortest accepting computation, and*
- *If there is no accepting computation of M on input x, then $T_M(x) = \infty$.*

Definition 2.9 (**Complexity classes**). *Let L be a language, a Turing machine M operates in time $f(n)$ where $f(n)$ is a function $f : \mathbb{N} \to \mathbb{N}$ if, for any input $x \in L$, $T_M(x) \leq f(|x|)$. We say that L belongs to time complexity class $DTIME(f(n))$ if L is decided by a deterministic Turing machine operating in time $f(n)$ analogously L is in time complexity class $NTIME(f(n))$ if L is decided by a non-deterministic Turing machine M operating in time $f(n)$. The space complexity class $s(n)$ is defined similarly where $s : \mathbb{N} \to \mathbb{N}$, and $s(n)$ in the total number of cells visited by the tape-head on an input of length n. Accordingly, a time complexity classes are sets of languages that can be decided within a certain bound. The definitions of some commonly used complexity classes and there names are:*

- *The class **P** consists of all languages whose yes/no solution can be obtained using a deterministic Turing machine whose running time is bounded above by a polynomial in n.*
- *The class **NP** consists of all languages whose yes/no solution can be obtained using a non-deterministic Turing machine whose running time is bounded above by a polynomial in n.*
- *The classes **PSPACE** and **NPSPACE** are defined analogously for deterministic and non-deterministic Turing machine with polynomial space bound.*

An additional concept is the concept of a *reduction* for establishing connections between different computational problems. Such connections are made by describing transformations from one problem to another and therefore permitting to develop decidability results from existing ones. A transformation is basically a function that maps strings of one problem into strings of another problem. Let $A \subseteq \Sigma^*$ and $B \subseteq \triangle^*$ be two arbitrary problems, which are encoded as sets of strings over the alphabets Σ and $\triangle$, respectively. A Turing-computable function $f : \Sigma \to \triangle$ is a transformation of A into B, if the following property hold:

$$\forall x \in \Sigma^* x \in A \Leftrightarrow f(x) \in B$$

That is, each input x can be algorithmically transformed into a word $f(x)$ such that x is an element of A if and only if $f(x)$ is an element of B.

Definition 2.10 (**Reducible**). *Let $A \subseteq \Sigma^*$ and $B \subseteq \triangle^*$ be two arbitrary sets of strings. If there is a transformation f from a problem A to a problem B, then we say that A is reducible to B, denoted by $A \propto B$.*

With the definition of reduction known decidability or undecidability results for a given problem can be carried over to new problems, if one can construct a suitable reduction.

Lemma 2.1. *Let $A \subseteq \Sigma^*$ and $B \subseteq \triangle^*$ be two arbitrary problems.*

1. *If A is undecidable and $A \propto B$, then B is undecidable, too.*

2. *If B is decidable and $A \propto B$, then A is decidable, too.*

Observe that no restrictions were imposed on the transformation function f besides being Turing-computable, in complexity theory we are mostly interested in reductions that use a certain amount of time or space. Most important classes of reductions are those that are computable in polynomial time and logarithmic space.

Definition 2.11 (**Polynomial-time reduction**). *Let $A \subseteq \Sigma^*$ and $B \subseteq \triangle^*$ be two arbitrary problems, which are encoded as sets of strings over the alphabets Σ and $\triangle$, respectively. Suppose that there is a transformation $f : \Sigma^* \to \triangle^*$. Then A is polynomial time reducible to B, if $f(x)$ can be computed in polynomial time. This is denoted by $A \propto_{poly} B$.*

Definition 2.12 (**Log-space reduction**). *Let $A \subseteq \Sigma^*$ and $B \subseteq \triangle^*$ be two arbitrary problems, which are encoded as sets of strings over the alphabets Σ and $\triangle$, respectively. Suppose that there is a transformation $f : \Sigma^* \to \triangle^*$. Then A is log-space reducible to B, if $f(x)$ can be computed using $\mathcal{O}(\log |x|)$ space. This is denoted by $A \propto_{log} B$.*

The term "NP-complete" signifies the subclass of decision problems in NP that are hardest in the sense that if one of them is shown to be solvable by a polynomial time deterministic Turing machine, then all problems in NP are solvable by a polynomial time deterministic Turing machine.

Definition 2.13 (**NP-Hardness**). *A decision problem B is NP-hard if for every problem A in NP, A* $\propto_{poly}$ *B holds.*

Definition 2.14 (**NP-Completeness**). *A decision problem B is said to be NP-complete if*

1. *B is in NP, and*

2. *for every problem A in NP, A* $\propto_{poly}$ *B, i.e. B is NP-hard.*

Consequently, the difference between an NP-complete problem B and an NP-hard problem A is that B must be in the class NP whereas A may not be in NP.

The definition of PSpace-hardness and PSpace-completeness is analogously to the definition of NP-hardness and NP-completeness.

2.3. Petri Nets

Petri nets (Petri (1962)) were first introduced by Carl Adam Petri, are formalisms to model and study concurrent system. As a general accepted formal models of concurrency they are option for studying the principles underlying different classes of systems. To

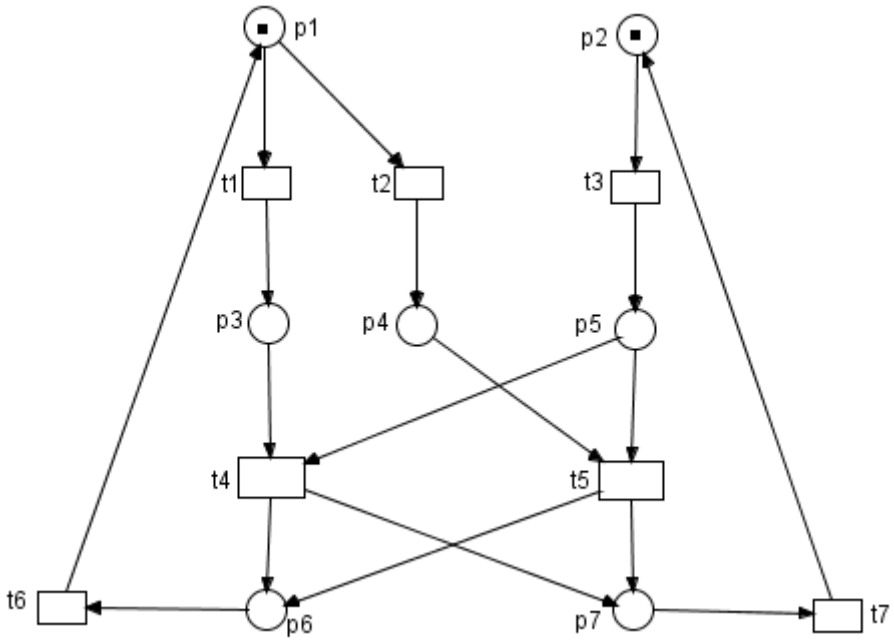

Figure 2.2.: A Petri net system

give the reader an intuitive knowledge of Petri net we illustrate the key features on a small example that modelled two communicating automata as Petri net taken from (Esparza and Heljanko (2008)). The states of the first automaton are p_1, p_3, p_4, p_6 and the states of the second are p_2, p_5, p_7. The automata synchronise on the transitions t_4, and t_5 . Figure 2.2 presents a graphical representation of the Petri net system. Basically, Petri nets are composed of three types of elements: *places*, depicted as circles, *transition*s, depicted as squares and *arcs* labelled with their weights (nonnegative integers), which must only link places to transition and vice versa, but never places to places or transitions to transitions. Places symbolises states, condition, or resources that need to be available before an action can happen. Transitions symbolises actions. Some places, like p_1 and p_2 in the figure are depicted with a black dot inside them. Such black dot are called *tokens*. Tokens may move to other places by executing actions. A token on a place means that corresponding condition is fulfilled or that a resource is available. The *preset* of a place is the set of those transitions from which there is an arc to the place; the *postset* of a place is the set of those transitions reached by an arc originating from the place. Likewise, the preset of a transition is the set of those places from which there is an arc to the transition; the post-set is the set of those places reached by an arc originating from the transition.

Petri nets can be investigated with respect to so called *sequential semantics* and *non-sequential semantics*. However, there is the third kind which can be derived from the other semantics by considering multiset of transitions that may be executed collectively called *step semantics*.

We start by introducing the notion of a marking for net systems under sequential semantics. A *state* or *marking* of the net is the distribution of tokens to places. The *initial marking* is the one frequently given with the net: in Figure 2.2 above, the initial marking is the marking that assigns one token to places p_1 and p_2 and none to the remaining places. From a marking a transition which is enabled may *fire* or *occur* and change the marking to a new marking. A transitions is said to be *enabled* at a marking if it marks each place in the preset of the transition. For example, in Figure 2.2 the initial marking enables, transitions t_1, t_2 and t_3, but does not enable t_4 or t_5 , as there are no tokens currently on places p_3, p_4 and p_5. An enabled transition may or may not fire depending on whether the event takes place. A firing of an enabled transition removes tokens in places of the preset of the transition and add tokens to the places in the postset. An *occurrence sequence* is any sequence of transitions that can occur from the initial marking in the order specified by the sequence, and the last marking it produces is said to be *reachable*.

Finally, the *reachability graph* which represent each net markings and the single transition firing between them, is a rooted, directed edge-labelled graph or simply a rooted *digraph* whose nodes are the reachable markings and such that there is an edge labelled by a transition t between markings m and m' if and only if t is enabled at m and after firing, produces the successor marking m'. It is clear to see that the firing sequences of the net system correspond to paths on the reachability graph starting from the initial marking. The reachability graph is frequently called the *state space* of the net system.

2.3.1. Formal Definition of Place/Transition Nets

The fundamental of all Petri net models is the definition of a net (Petri (1966)).

Definition 2.15 (P/T net). *A place/transition (P/T net for short) is a tuple* $N = (P, T, F, W)$ *where*

- P *is a set of places,*
- T *is a set of transitions, disjoint from* P*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is called the flow relation (The flow relation is the set of arcs), and*
- $W : F \longrightarrow \mathbb{N} \setminus \{0\}$ *is the arc weight function.*

If P and T are finite, the net N is said to be finite

If $W(x,y) = 1$ for all arcs $(x,y) \in F$ we usually omit W in the tuple and simply write (P,T,F) for a P/T net. The preset of a node $x \in P \cup T$, denoted $\bullet x$, is the set containing the elements that immediately precede x in the net i.e. $\bullet x = \{y \in P \cup T \mid (y,x) \in F\}$ Likewise, the postset of a node, denoted $x\bullet$, can be defined. Given a set $X \subseteq P \cup T$, the notion of preset/postset can be extended to $X : X\bullet = \{y \mid x \in X, y \in x\bullet\}$. A P/T net system Σ is a (N, M_0) comprising a finite net $N = (P,T,F)$, together with an initial marking M_0, which is a function from set of places to the set of natural numbers, denoted as $M_0 : P \longrightarrow N$. For example Figure 2.4 on page 31. shows the graphical representation of the Petri net $N_1 = (P,T,F,W,M_0)$ where

- $P = \{p_1, p_2, p_3, p_4, p_5\}$,
- $T = \{t_1, \ldots, t_5\}$,
- $F = \{(p_1,t_1), (t_1,p_2), (p_2,t_2), (p_2,t_4), \ldots\}$ *and*
- $(p_1,t_1) = 1, W(t_1,p_2) = 1, \ldots, W(p_2,t_3) = 2 \ldots$
- $M_0 = p_1, p_4$

We have for instance $\bullet t_4 = \{p_3, p_4\}$ and $t_2\bullet = \{p_1, p_5\}$.

In an alternative definition of P/T net by (Girault and Valk, 2013) arcs are given by the backward and forward incident matrices **pre** and **post**. This notation is usually rather helpful when calculating invariants, whereas the usage of F is closer to the graphical representation

Definition 2.16 (P/T net (alternative)). *A place/transition (P/T net) is a tuple $N = (P, T, \boldsymbol{pre}, \boldsymbol{post})$ where*

- *P is a finite set of places,*
- *T is a finite set(the set of transitions of N), disjoint from P and,*
- *$\boldsymbol{pre}, \boldsymbol{post} \in \mathbb{N}^{|\mathbb{P}| \times |\mathbb{T}|}$ are matrices (backward and forward incidence matrices of N).*

$C = \boldsymbol{pre} - \boldsymbol{post}$ is called the incidence matrix of N.

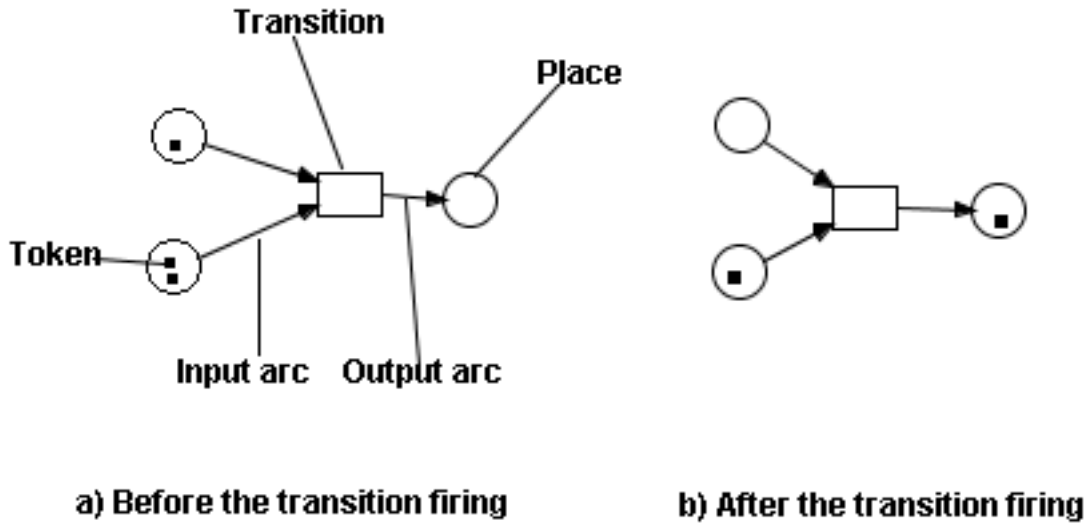

Figure 2.3.: Example of Petri net and transition firing

If there is an arc with weight $n > 0$ from a place $p \in P$ to some transition $t \in T$, then we have $pre[p,t] = n$ or alternatively $(p,t) \in F$ and $W(x,y) = n$. Similarly for arcs from transitions to places. The two representations can therefore easily be converted into each other. The behaviour of many systems can be described in term of system state and their changes. In order to simulate the dynamic behaviour of a system, a state or marking in a Petri net is changed according to the following transition firing rule:

- A transition t is said to enabled if each input place p of t has at least $F(p,t)$ tokens
- An enabled transition may or may not *fire* (depending on whether or not the event actually takes place)
- A firing of an enabled transition t removes $F(p,t)$ tokens from each input place p of t, and adds $F(t,p)$ tokens to each output place p of t.

Definition 2.17 (Marking). *A marking of a P/T net $N = (P,T,F,W)$ is a vector $m \in N^{|P|}$ or alternatively a function $M : P \longrightarrow N$.*

The special marking with $M(p) = 0$ for all $p \in P$ is denoted by 0, which is also used in the special case where N has no places, i.e. $P = \phi$.

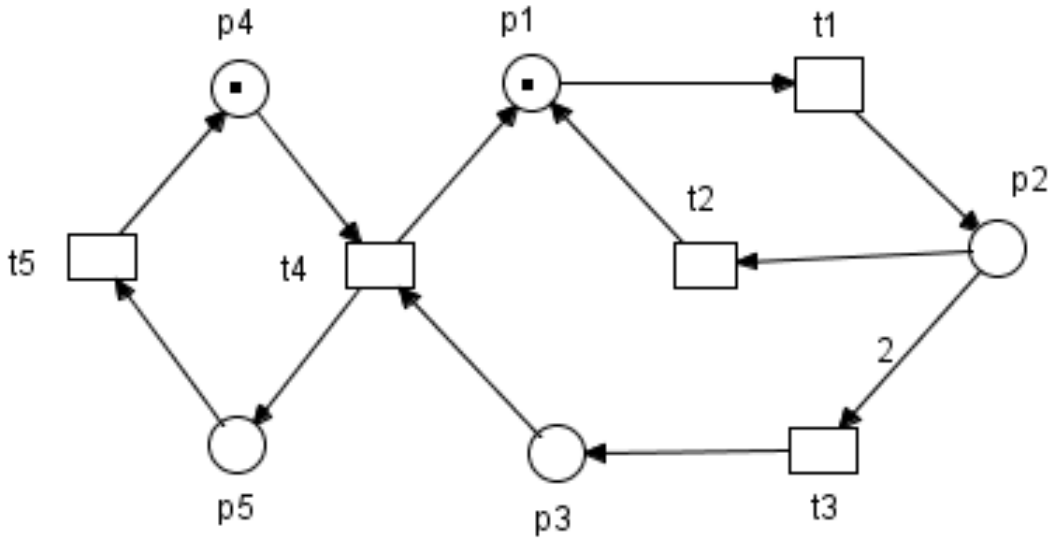

Figure 2.4.: A Petri net N_1

Definition 2.18. ***[Enabled transition]*** *Let $\Lambda = (N, m_0)$ be a net system. A transition $t \in T$ is enabled in a marking M iff $M(p) \geq F(p,t)$ for all $p \in \bullet t$. Enabling of a transition t in a marking M is denoted by $M[t\rangle$.*

A transition that is enabled may fire. Firing removes the necessary amount of tokens from all places in the preset of t and puts new tokens on all places in postset of t according to the arc weight function. *A transition t that is enabled in a marking M i.e. $M[t\rangle$, may fire. The successor marking M' is defined as $M' = M + F(p,t) - F(t,p)$. We denote this by $M[t\rangle M'$.*

The Figure 2.3 on page 30 shows a Petri net before and after firing of a transition. Figure 2.3 (a) shows the marking before firing the transition and Figure 2.3 (b) shows the marking reached after firing the transition.

The set of reachable markings of Σ is the smallest ($w.r.t.$ $\subseteq$) set $RM(\Sigma)$ containing M_0 and such that if $M \in RM(\Sigma)$ and $M[t\rangle M$, for some $t \in T$ and $M' \in RM(N)$, then $M' \in RM(\Sigma)$. For finite sequence of transition $\sigma = t_1, \ldots, t_k$, we write $M[\sigma\rangle M'$ if there are markings $M_1, \ldots, M_{k+1}$ such that $M_1 = M$, $M_{k+1} = M'$ and $M_i[t_i\rangle M_{i+1}$, for all $i = 1, \ldots, k$.

A marking is deadlocked if it does not enable any transitions. Σ is deadlock free if none of

its reachable markings is deadlocked. A transition t is dead if no reachable marking enables it. M covers M, if $M \leq M'$. A marking M is coverable if there exists $M' \in RM(\Sigma)$ such that M covers M.

Σ is k-bounded if, for every reachable marking M and every place $p \in P$, $M(p) \leq k$, and safe if it is 1-bounded. Moreover, Σ is bounded if it is k-bounded. For some $k \in N$. One can show that the set $RM(\Sigma)$ is finite if Σ is bounded i.e. if $|RM(\Sigma)| < \infty$.

The definition of 1-safe nets can be seen a particular case of the definition of P/T nets where all arcs have a weight of 1, and there is at most one token in each place in every reachable marking. Albeit P/T nets and 1-safe nets have similar graphical and mathematical representations they are rather different: for instance, finite P/T net can have infinite state space, but finite 1-safe nets cannot.

We now define standard problems in the context of Petri nets and verification in general. The *reachability* problem asks if a given marking is reachable in the net system. The *liveness* problem asks if every transition can always be enabled again. The *coverability* problem asks if a marking M can at least be *covered by a reachable marking M', i.e if M' with $M' \geq M$is reachable. The boundedness problem asks if the set of reachable markings is finite.*

Definition 2.19 (**Petri net decision problems**). *1. In the reachability problem a P/T net system $\Sigma = (N, m_0)$ and a marking m of Σ are given and the question is, if $m_0[*\rangle m$ holds.*

2. *In the liveness problem a P/T net system $\Sigma = (N, m_0)$ is given and the question is, if Σ is live.*

3. *In the coverability problem a P/T net system $\Sigma = (N, m_0)$ and a marking m of Σ are given the question is, if m' exist such that $m' \in RM(\Sigma)$ and $m' \geq m$.*

4. *In the boundedness problem a P/T net system $\Sigma = (N, m_0)$ is given the question is, if Σ is bounded.*

All the problems mentioned above are decidable for P/T nets, but to solve them in the general case requires at least exponential space and thus the complexity is rather high. Indeed most interesting questions about P/T nets are EXPSPACE-Hard (see for example, the survey articles by Esparza and Silva (1992) and Esparza and Nielsen (1994)). Note that EXPSPACE-Hardness does not mean that these problems are decidable. In-

deed many important problems for P/T nets are actually undecidable, among them the model checking problems for LTL and CTL and equivalence problems (see for instance Esparza and Silva (1992)). However, it has been proven in the literature that the reachability, liveness, coverability, and boundedness problem are decidable for P/T nets, but are EXPSPACE-Hard. Coverability and boundedness can be decided in EXPSPACE.

2.4. Petri Net Markup Language (PNML)

The *Petri Net Markup Language* is XML-based interchange format for Petri nets. It is aimed at enabling Petri net tools to exchange Petri net models. Also, for real-life applications, PNML as an XML transfer format for Petri nets can be parsed into a programming language to enable the user do something useful with it. For instance, there is the SAX (Simple API for XML) called the JAXP which is used to parse PNML as XML into a Java program and then manipulate it to suite your needs. We will require PNML representation of Elementary Reference-net System later in chapter 5, which will be used as input language for the implementation of the software tool to be developed solely for transforming ERS into low-level Petri net. More detailed explanation on PNML syntax, and the semantics in particular, can be found in (Weber et al., 2003). For readers who are interested in the concept, technology and tool for standard PNML, the book by (Billington et al., 2003) gives a detailed account of such topics

2.5. RENEW

The Reference Net Workshop RENEW (Kummer, 2002) is a Petri net editor and simulator specially developed for modelling based on nets-within-nets. It is described in Kummer et al. (2004) and the recent documentation is contained in Cabac et al. (2015). It provides all necessary facilities required to create, edit, simulate and inspect Object-oriented Petri Nets, High-level Petri Nets, Place/Transition Nets and Petri Nets with Time. Although it is not restricted to these formalisms. It has served as the build- and run-time environment for many software systems using reference nets (see Wagner (2010). More importantly, RENEW contains mechanisms that allows interchange of PNML format for nets. It is possible to design a net graphically and export it's PNML representation to other applications. Renew is free of charge and available online. It is maintained by the Theoretical Foundations Group of the Department for Informatics of the University of

Hamburg Germany. As RENEW supports the idea of nets-within-nets and it is possible to design a net graphically and export the PNML format for use with other applications, and also, the PNML functionality allows multiple nets to be contained within one PNML file, we have chosen it as the basic environment for the creation, modification and developing the ERS and it's PNML representation. In particular, in Chapter 5, a PNML net representation of ERS which is used as an input language for the implementation of the our developed tool for transforming ERS into P/T net will be discussed.

Chapter 3.

Basic Elementary Object Systems

3.1. Informal Introduction and Motivation

In this section we present an introductory example two- level nesting hierarchy of *nets-within-nets* (NWN) formalism first introduced by Valk (1998) generally referred to as Object Petri nets. This formalism provides a modelling technique that allows tokens of Petri nets to be Petri nets themselves, called *token nets* or *object-nets* and have been studied with respect to two semantics: reference and value semantics. Interactions between the surrounding net called *system-net* and object-nets are done by *synchronisation* of transitions. Some transitions in the system net and token nets that must fire synchronously are labelled by a corresponding synchronisation channels.

The behaviour of object systems include three kinds of events: a *system-autonomous event*, an *object-autonomous event* and a *synchronous event*. In a system-autonomous event, a transition can *move* or *remove* object-nets but does not change their internal states. An object-autonomous event changes only an inner state in object net it belongs to. In a synchronous event the system nets transition fires synchronously together with a corresponding transition in the object net.

To give an intuitive understanding of Elementary Object net Systems (EOS) we apply the key features to a small example of an imagery agent that moves around in some environment such as a Library. The agent moves from the hall to a reading room picks up some books and transports them to the book-store where they are kept on shelves. An elementary object net modelling this system could be that shown in Figure 3.1 on page 36. The system consist of a system net SN, modelling the environment with tokens of two types: black tokens, and an agent net represented by ordinary Petri net N1 called

the object net initially on the place p_3 - indicated by a ZOOM. The places p_2, p_3, and p_4 in the system net describes the reading-room, the hall, and the book-store of the scenario and are typed with N1, meaning that on these places only object net of this agent net type

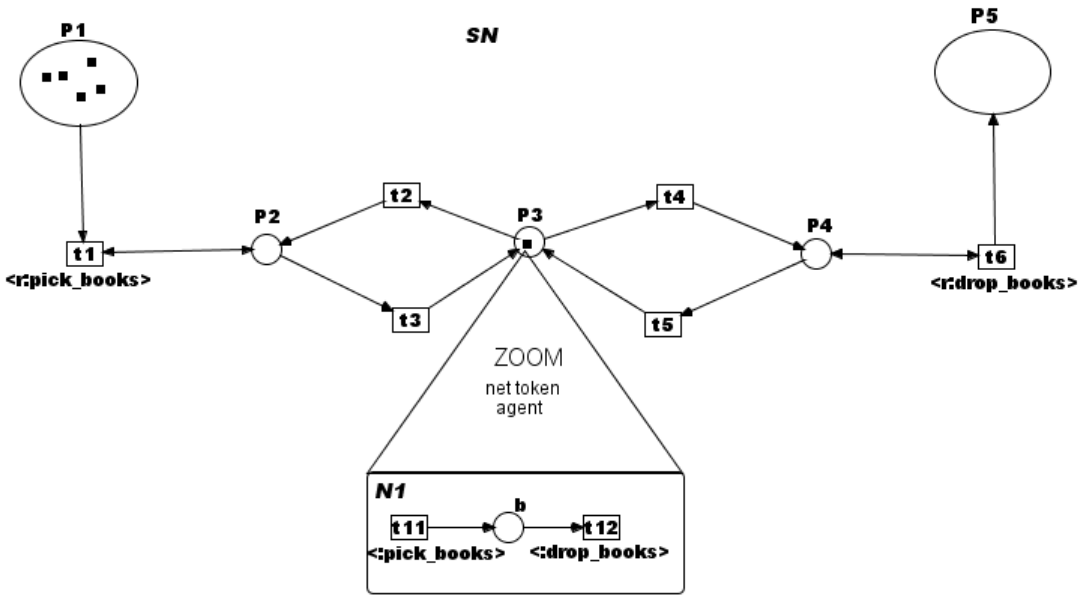

Figure 3.1.: Elementary Object Net - EOS1

N1 may reside, and the transitions t_2, t_3, t_4, and t_5 are the movement of the agent between them. The other places p_1, and p_5 are typed with black token representing books, and the transitions t_1 and t_6 are the actions for picking and dropping of the books picked previously. We note that the agent can pick up and returns a book. A Petri net that models this behaviour is depicted as N1 in the figure. System nets and object nets synchronise via channels. The channels are denoted as labels in angular brackets such as $< r : pick\_books >$ in the system-net and $<: pick\_books >$ in the net token. Transitions with corresponding channels must fire synchronously. Transitions without a labels can fire autonomously and concurrently to other enabled transitions.

Let us now describe the dynamic behaviour of the system. Initially the scenario is as depicted in Figure 3.1 on page 36, where the agent modelled with the Petri net N1 resides in place p_3 that models the hall of the library. Thus we do not have a black token but a Petri net on p_3. The marking of the agent is the empty making 0 since initially it has not picked up any book. Marking of ERS are describe via nested multisets. Therefore, we begin with the marking

$$p_3[0] + 5'p_1$$

meaning that p_3 is marked with an object net whose marking is 0 and the place p_1 is marked with five black tokens representing books. The system-autonomous event t_2 which models the movement of the agent to the reading-room p_2, is activated and can fire resulting in the successor marking

$$p_2[0] + 5'p_1$$

Since the agent has not yet picked a book, the marking remains the empty marking 0. The synchronous events consisting of transition t_1 in the system-net and t_{11} in the agent net denoted by $t_1[t_{11}]$ is now activated, both transitions are activated and can fire and leads to the successor marking

$$p_2[b] + 4'p_1$$

The system-autonomous event t_3 which models the movement of the agent back to the hall is activated and can fire resulting to successor marking

$$p_3[b] + 4'p_1$$

The system-autonomous event t_4 which models the movement of the agent to the book-store p_4, is activated and can fire resulting in the successor marking

$$p_4[b] + 4'p_1$$

When on place p_4 the agent drops the book from p_1 by firing the synchronous events $t_6[t_{12}]$. Note that due to the typing of the places, the net token $N1$ is placed on p_4 again and not on p_5. Place p_5 is marked with black token, and the resulting marking is

$$p_4[0] + 4'p_1 + 1'p_5$$

The agent can then return to the hall: place p_3, via the system-autonomous event t_6 resulting in

$$p_3[0] + 4'p_1 + 1'p_5$$

Continuing in this manner, the agent can transport all the books from the reading-room to the book-store.

Many additions to the model are imaginable, for example, extending the model of agent with a counter place would allow the agent to carry multiple books. We hope, the rough model in this example is sufficient to give a reader an intuitive understanding of the main points of the formalism.

Figure 3.1 (on page 36) shows one of reachable states of EOS1, where the agent has transferred a book from the reading-room to the book-store and have returned to its initial position p_3.

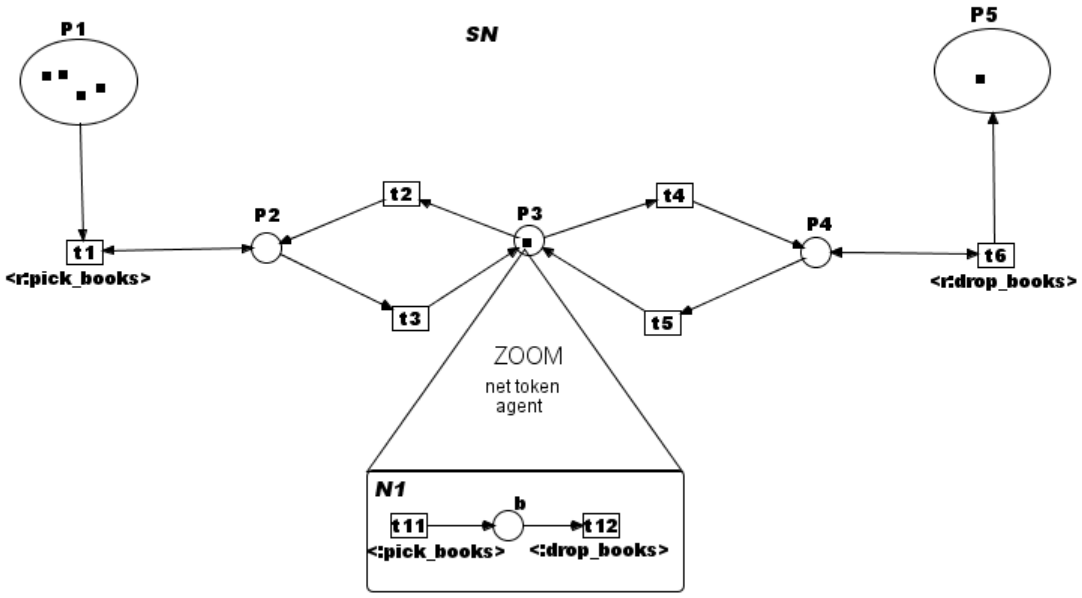

Figure 3.2.: An example of a reachable state for EOS1

Several works on Elementary Object Systems have been studied in the literature. One most important result in Heitmann (2013) shows that under some dynamic restrictions, the size of the reachability graph for EOS is finite, and every property that can be expressed in the temporal logics LTL or CTL can be verified in polynomial space. This result, not only shows that EOS are well rooted inside the theory of Petri nets, it also allows us to carry over properties of the model one can specify on the 1-safe Petri nets by means of techniques developed in the literature for their analysis on elementary object nets. Another result in Heitmann and Köhler-Bußmeier (2012), introduces a *logical language* for reasoning about systems representable with EOS. The language combines two families of modal operators: one family cope with the evolution of the described system in *time*, the other deals with *spatial configurations*, i.e. a logic in which we can express for example, that a certain object or agent is always somewhere or at a specific location. The problem of model checking properties of a class of logic on this dynamically restricted EOS was also shown to be **PSPACE-complete**.

3.1.1. Formal Definition of EOS

In this section we recapitulate and summarised the formal definitions of elementary object system framework as presented in Heitmann (2013) for the scenario of an EOS we intuitively described in the previous section.

3.1.2. The static structure of EOS

As informally described in the previous section, an elementary object system is composed of a system net, a set of object nets, a typing of the system net places and a labelling of the transitions with channels or with the special symbol τ. The labelling has to adhere to some restrictions and from the labelling the set of events can be deduced. An example is given in Figure 3.3 on page 40, see also Example 3.2 below

Definition 3.1 (**EOS**). *Let C be a set of synchronisation channels and $\tau \notin C$. An elementary object system is a tuple $OS = (\widehat{N}, \mathcal{N}, d, l)$ such that:*

1. *$\widehat{N} = (\widehat{P}, \widehat{T}, pre, post)$ is a P/T net, called the system.*

2. *$\mathcal{N} = N_1, \ldots, N_n$ is a finite set of disjoint p/t nets called object nets given as $N_i = (P_{N_i}, T_{N_i}, pre_{N_i}, post_{N_i})$.*

3. *$d : \widehat{P} \longrightarrow \mathcal{N}$ is a typing function of the system net places.*

4. *$l = (\widehat{l}, (l_N)_{N \in \mathcal{N}})$ is the labelling in which*

$$\widehat{l} : \widehat{T} \longrightarrow (\mathcal{N} \longrightarrow C \cup \{\tau\}) \text{ and}$$

$$l_N : T_N \longrightarrow (C \cup \{\tau\}) \text{ for all } N \in \mathcal{N}$$

.

It is assumed that $N \in \mathcal{N}$ and the object net $N_\bullet \in N$, which has no places or transitions and is used to model black tokens. Moreover, it is assumed that all sets of nodes (places and transitions) are pairwise disjoint and set $P_{\mathcal{N}} := \cup_{N \in \mathcal{N}} P_N$ and $T_{\mathcal{N}} := \cup_{N \in \mathcal{N}} T_N$.

The system net places are typed by the mapping $d : \widehat{P} \longrightarrow N$ with the meaning that $d(\widehat{P}) = N$, then the place $\widehat{P}$ of the system net can contain only net-token of the object

net type N. The transitions in an EOS are labelled with synchronisation channels by the synchronisation labelling l. For this, we use a fixed set of channels C. In addition it is allowed that the label τ is used to denote that no synchronisation is desired (i.e. autonomous firing). The intended meaning of the labels is as follows:

1. $l_N(t) = \tau$ means that the transition t of the object net N, is without synchronisation label and may fire object-autonomously.

2. $l_N(t) = c \neq \tau$ means that the transition t of the object net N, has a synchronisation label and may fire synchronously via the channel c with the system net.

3. $\widehat{l}(\widehat{t})(N) = \tau$ means that the system net transition $\widehat{t}$ is without synchronisation label and may fire autonomously from the object net. If $\widehat{l}(\widehat{t})(N) = \tau$ holds for every $N \in N$ then he system net transition $\widehat{t}$ may fire system-autonomously.

4. $\widehat{l}(\widehat{t})(N) = c \neq \tau$ means that $\widehat{t}$ has synchronisation label and may fire synchronously via the channel c with the object net N.

In case of an autonomous event a single transition fires independently from all other transitions. In case of a synchronous event a system net transition fires together with a system net net transition also labelled with c. In this case the labels i.e. the channels used have to match. (See Definition 3.3 on page 42).

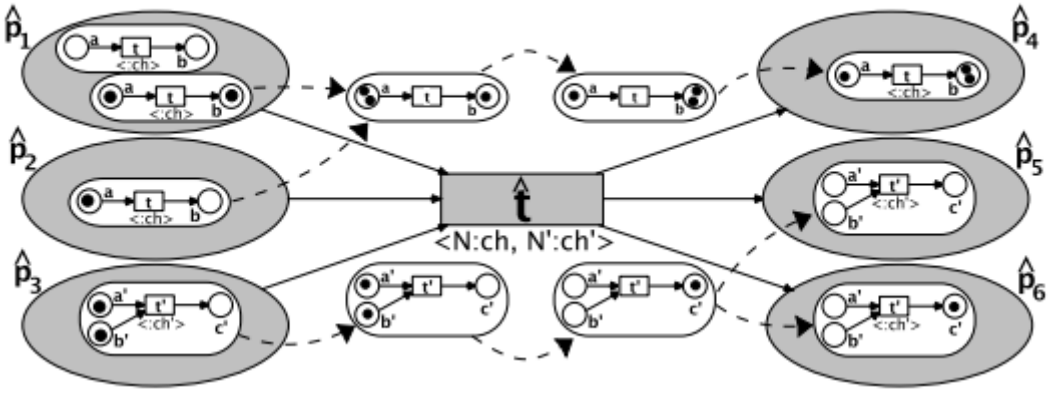

Figure 3.3.: An EOS firing the synchronous event $\widehat{t}[t, t']$.

Definition 3.2 (**Marking of an EOS**). *A marking of an EOS, OS is a nested multiset. Let*

$$M_{\aleph} = \bigcup_{\widehat{p} \in \widehat{\mathbf{P}}} (\{\widehat{P}\} \times \mathcal{M}_{d(\widehat{p})})$$

A marking of OS is a finite multiset $\mu : M_N \longrightarrow \mathbb{N}$.

We will usually denote an element $(\widehat{p}, M)$ *of* $M_{\mathcal{N}}$ *by* $\widehat{p}[M]$ *and a marking of an EOS by* $\mu = \Sigma_{k=1}^{|\mu|} \widehat{p}_{ik}[M_{ik}]$, *where* $\widehat{p}_{ik}$ *is a place in the system net,* M_{ik} *is a marking of a net token of type* $d(\widehat{p}_k)$ *and* i_k *are indices. With* $|\mu|$ *we denote net-tokens present in the number of elements from* $M_{\mathcal{N}}$ *that appear in* μ, *i.e. the number* $\Sigma_{(\widehat{p},m)\in M_{\mathcal{N}}}\mu(\widehat{p}, m)$, *which is the number of net-tokens present in* μ. *As a shortcut we write the sum also as* $\mu = \Sigma_k \widehat{p}_k[M_k]$.

We define the partial order on nested multiset by setting $\mu_1 \leq \mu_2$ *iff* $|\mu_1| \leq |\mu_2|$ *and the elements of* μ_1 *and* μ_2 *can be arranged in such a way that* $\mu_1 = \Sigma_i \widehat{p}_i[M_i]$, *and* $\mu_2 = \Sigma_j \widehat{p'}_j[M'_j]$ *and for all* $k \leq |\mu_1|$ *we have* $\widehat{p}_k = \widehat{p'}_k$ *and* $M_k \leq M'_k$, *where* $\leq$ *is the usual multiset relation. Thus, the same net-tokens appear in* μ_2 *with at least the same marking as in* μ_1 *and in* μ_2 *may appear additional net-tokens*

With $\preceq$ *we denoted the special partial order with* $\mu_1 \preceq \mu_2$ *iff* $|\mu_1| \leq |\mu_2|$ *and the elements of* μ_1 *and* μ_2 *can be arranged in such a way that* $\mu_1 = \Sigma_i \widehat{p}_i[M_i]$, *and* $\mu_2 = \Sigma_j \widehat{p'}_j[M'_j]$ *and for all* $k \leq |\mu_1|$ *we have* $\widehat{p}_k = \widehat{p'}_k$ *and* $M_k \leq M'_k$. *Thus every net-token that appears in* μ_1 *appears in* μ_2 *and with the same marking. In* μ_2 *additional net-tokens may appear.*

The set of all markings of OS is denoted by $\mathcal{M}_{OS}$ *or simply* $\mathcal{M}$ *if no ambiguity can arise.*

An EOS with initial marking is a tuple $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *where* $\mu_0 \in \mathcal{M}$ *is the initial marking.*

In the above definition an element of M_N is a marking of an object net together with the place in the system net on which it resides. A marking μ of OS then gives the position of all net tokens and their inner marking. Multiplicity is only needed (in μ) if more than one net token with the same marking resides on the same place of the system net.

3.1.3. The dynamic behaviour

To introduce the firing rule for EOS we need to introduce events, which correspond to transitions in a p/t net, projections and the enabling predicate. In the following definitions we usually assume that an Eos as in Definition 3.1 (a) on page 39 is given.

The set of system events Θ is generated by the synchronisation labelling. The set Θ consists of the disjoint sets of synchronous events Θ_l, object-autonomous events Θ_o, and

system-autonomous events Θ_s. An event is a pair, denoted by $\widehat{t}[\vartheta]$ in the following, where $\widehat{t}$ is a transition of the system net or $\widehat{\epsilon}$ and ϑ maps each object net to one of its transitions or to ϵ. The mapping has to be consistent with the labelling and in a synchronous event the labels of the participating transitions have to match. The symbol $\widehat{\epsilon}$ is used for object-autonomous firing. The special mapping that maps each object net to $\widehat{\epsilon}$ i.e. $\vartheta(N) = \epsilon$ for all N is denoted by ϑ_ϵ. In such an event the system net transition fires autonomously

Definition 3.3 (**Event**). *An event is a pair denoted $\widehat{t}[\vartheta]$ where $\widehat{t}$ is a transition of the system net or the special symbol $\widehat{\epsilon}$ and $\vartheta : N \longrightarrow T_{\mathcal{N}} \cup \{\epsilon\}$ is a mapping where $\vartheta(N) \neq \epsilon$ implies $\vartheta(N) \in T_N$ for all $N \in \mathcal{N}$.*

The labelling functions are extended to $l_N(\epsilon) = \tau$ and $\widehat{l}(\widehat{\epsilon}) = \tau$ for all $N \in \mathcal{N}$.

There are three possible kinds of events:

1. *In a synchronous event the system net transition $\widehat{t} \neq \widehat{\epsilon}$, fires synchronously with all the object net transitions $\vartheta(N), N \in \mathcal{N}$. At least one $N \in \mathcal{N}$ must exist with $\widehat{l}(\widehat{\epsilon})(N) \neq \tau$ and $\vartheta(N) \neq \epsilon$. We require $\vartheta(N) \neq \epsilon \Leftrightarrow \widehat{l}(\widehat{t})(N) \neq \tau$ and that the channels have to match, i.e. $\widehat{l}(\widehat{t})(N) = l_N((\vartheta(N))$ for all $N \in \mathcal{N}$.*

2. *In a system-autonomous event $\widehat{t} \neq \epsilon$ fires autonomously. We require that $\widehat{l}(\widehat{t})(N) = \tau$ for all $N \in \mathcal{N}$ and $\vartheta(N) = \epsilon$ for all $\mathcal{N}$, i.e. $\vartheta = \vartheta_\epsilon$.*

3. *In an object-autonomous event we require $\vartheta(N) \neq \epsilon$ for exactly one object net N. Moreover, the transition $\vartheta(N)$ must not use a channel, that is $l_N(\vartheta(N)) = \tau$ has to hold.*

The set of synchronous events is denoted by Θ_l, the set of system-autonomous events is denoted by Θ_s, and the set of object-autonomous events is denoted by Θ_o. The set of (system) events is the disjoint union of these three sets:

$$\Theta := \Theta_l \cup \Theta_s \cup \Theta_o$$

Note that for object nets which do not participate in a synchronous event (either because they are not in the preset of the system net transition or because no object net transitions fires synchronously) $\widehat{l}(\widehat{t})(N) = \tau$ holds, which forces $\vartheta(N) = \epsilon$ and thus $l_N(\vartheta(N)) = l_N(\widehat{\epsilon}) = \tau = \widehat{l}(\widehat{t})(N)$.

The requirements for a system-autonomous event imply $\vartheta(N) \neq \epsilon \Leftrightarrow \widehat{l}(\widehat{t})(N) \neq \tau$, the

equivalence we had to demand in the case of a synchronous event. Moreover,$l_N(\vartheta(N)) = \widehat{l}(\widehat{t})(N) = \tau =$ follows, too.

Also in an object-autonomous event the labels match again for all N, i.e.$\widehat{l}(\widehat{t})(N)) = = l_N(\vartheta(N))$ for all $N \in \mathcal{N}$, but the equivalence $\vartheta(N) \neq \epsilon \widehat{l}(\widehat{t})(N) \neq \tau$ does not hold for exactly one N, namely for the N for which $\vartheta(N) \neq \epsilon$ holds. $\vartheta(N) \in T_N$ is the transition intended to fire object-autonomously.

If we write $\widehat{t}[\vartheta] \in \Theta$ in the following, this includes the possibility that the event is a system- or an object-autonomous event, i.e. $\vartheta = \vartheta_\epsilon$ or $\widehat{t} = (\epsilon)$ is possible. Moreover, since the sets of transitions are all disjoint, we usually write $\widehat{t}[\vartheta N_1, \vartheta N_2, \ldots]$ and also skip the object nets which are mapped to ϵ, that is, we simply list the object nets transitions with which a system net transition synchronises.

Example 3.1. Figure 3.3 on page 40 shows an EOS consisting of a system net $\widehat{N}$ and two object nets $\mathcal{N} = \{N, N'\}$. The typing of the system net is given by $d(\widehat{p}_1) = d(\widehat{p}_2) = d(\widehat{p}_4) = N$ and $d((\widehat{p}_3) = d(\widehat{p}_5) = d(\widehat{p}_6) = N'$. For now, ignore the net-tokens on $\widehat{p}_4, \widehat{p}_5$ and $\widehat{p}_6$. These places are initially empty and the system has thus four net-tokens: two on place $\widehat{p}_1$ and one on $\widehat{p}_2$ and $\widehat{p}_3$ each. The net-tokens on $\widehat{p}_1$ and $\widehat{p}_2$ share the same structure, but have independent markings. The initial marking is thus given by

$$\mu = \widehat{p}_1[0] + \widehat{p}_1[a+b] + \widehat{p}_2[a] + \widehat{p}_3[a+b].$$

We have two channels ch and ch′. The labelling function $\widehat{l}$ of the system net is defined by $\widehat{l}(\widehat{t})(N) = ch'$ and $\widehat{l}(\widehat{t})(N')) = ch'$. The object nets labellings are defined by $\widehat{l}_N(t) = ch$ and $\widehat{l}_{N'}(t') = ch'$. Thus there is only one (synchronous) event: $\Theta = \Theta_l = \{t[N \mapsto t, N' \mapsto t']\}$. The event is also written shortly as $\widehat{t}[t, t']$.

Projections are needed e.g. ignoring the inner markings of net tokens in case of a system-autonomous event.

Definition 3.4 (**Projections**). *Let $\mu \in \mathcal{M}_{\mathcal{OS}}$ be a nested marking of an EOS OS. $\Pi^1(\mu)$ denotes the projection of the nested marking μ to the system net level and $\Pi^2_N(\mu)$ denotes the projection to the marking belonging to the object net N, i.e.*

$$\Pi^1(\sum_k \widehat{p}_k[M_k]) = \sum_k \widehat{p}_k \text{ and}$$

$$\Pi^2_N(\sum_k \widehat{p}_k[M_k]) = \sum_k 1_N(\widehat{p}_k).M_k$$

where $1_N : \widehat{P} \longrightarrow \{0,1\}$ *with* $1_N(\widehat{p}) = 1$ *iff* $d(\widehat{p}) = N$.

$\Pi^1(\mu)$ is again a multiset i.e. a function $\widehat{P} :\longrightarrow \mathbb{N}$ such that $\Pi^1(\mu)(\widehat{p})$ is the number of net-tokens that reside on $\widehat{p}$, but can also be seen as a vector from $N^{|\widehat{P}|}$. Similarly Π^2_N is also a multiset, it is the marking of the net N viewed as a p/t net. Note, however, that Π^2_N adds up all markings of object nets of type N regardless to where these object net reside in the system net. Π^2_N can also be seen as a vector from $\mathbb{N}^{|p_N|}$.

To explain firing we distinguish two cases: Firing a system-autonomous or synchronous event $\widehat{t}[\vartheta] \in \Theta_l \cup \Theta_s$ removes net-tokens together with their individual internal markings. The new net-tokens are placed according to the system net transition and the new internal markings are determined by the internal markings just removed and ϑ. Thus a nested multiset $\lambda \in \mathcal{M}$ that is part of the current marking μ, i.e. $\lambda \preceq \mu$, is replaced by a nested multiset ρ. The marking μ is not needed to define the enabling predicate, but is needed in the firing rule below.

Definition 3.5 (**Enabling predicate**). *Let OS be an EOS and* $\lambda, \rho \in M$ *be markings. Let* $\widehat{t}[\vartheta] \in \Theta$ *be an event. The enabling condition is expressed by the enabling predicate* ϕ_{OS} *(or just* ϕ *whenever OS is clear from the context). We distinguish two cases:*

1. $\widehat{t}[\vartheta] \in \Theta_l \cup \Theta_s$ *is a synchronous or system-autonomous event. In this case we have* $\phi(\widehat{t}[\vartheta], \lambda, \rho)$ *if and only if*

$$\begin{aligned}\Pi^1(\lambda) = pre(\widehat{t}) \wedge \Pi^1(\rho) = post(\widehat{t})\\ \wedge \forall N \in \mathcal{N} : \Pi^2_N(\lambda) \geq pre_N(\vartheta(N)\\ \wedge \forall N \in \mathcal{N} : \Pi^2_N(\rho) = \Pi^2_N(\lambda) - pre_N(\vartheta(N)) + post_N(\vartheta(N))\end{aligned}$$

   where $pre_N(\epsilon) = post_N(\epsilon) = 0$ *for all* $N \in \mathcal{N}$.

2. $\widehat{\epsilon}[\vartheta] \in \Theta_o$ *is an object-autonomous event. In this case let* N *be the object net for which* $\vartheta(N) \neq \epsilon$ *holds. Now* $\phi(\widehat{\epsilon}[\vartheta], \lambda, \rho)$ *holds iff* $\Pi^1(\lambda) = \Pi^1(\rho) = \widehat{p}$ *for a* $\widehat{p} \in \widehat{P}$ *with* $d(\widehat{p}) = N$ *and* $\Pi^2_N(\lambda) \geq pre_N(\vartheta(N))$ *and* $\Pi^2_N(\rho) = \Pi^2_N(\lambda) - pre_N(\vartheta(N)) + post_N(\vartheta(N))$.

In case of an object-autonomous event λ and ρ are thus essentially markings of an object net, but preceded by a system net place typed with this object net. Note that, in general, the event alone does not fully characterize the firing. For example, if an object net transition t fires autonomously, the mode λ is necessary, to describe where the object

net resides. This is especially important, if two object nets of the same type exist on different system net places.

We are now ready to define the firing rule.

Definition 3.6 (**Firing Rule**). *Let OS be an EOS and $\mu, \mu' \in \mathcal{M}$ markings.*

The event $\widehat{t}[\vartheta] \in \Theta$ is enabled in μ for the mode $(\lambda, \rho) \in \mathcal{M}^2$, denoted by $\mu \xrightarrow{\widehat{t}[\vartheta],\lambda,\rho}$ iff $\lambda \leq \mu \wedge \phi(\widehat{t}[\vartheta], \lambda, \rho)$ holds.

An event $\widehat{t}[\vartheta]$ that is enabled in μ for the mode (λ, ρ) can fire: $\mu \xrightarrow{\widehat{t}[\vartheta],\lambda,\rho} \mu'$. The resulting successor marking is defined as $\mu' = \mu - \lambda + \rho$.

As usual firing is extended to sequences $w \in (\Theta \cdot \mathcal{M}^2)^*$ inductively on the length of w

- $\mu \xrightarrow{\epsilon} \mu$
- *If $w = w' \cdot \Theta$ with $w' \in (\Theta \cdot \mathcal{M}^2)^*$ and $\Theta \in (\Theta \cdot \mathcal{M}^2)$, then $\mu \xrightarrow{w} \mu'$ iff a marking μ'' exists such that $\mu \xrightarrow{w'} \mu''$ and $\mu'' \xrightarrow{\Theta} \mu'$ holds.*

To denote that μ' is reachable from μ by some sequence of transitions we write $\mu \xrightarrow{*} \mu'$.

The set of reachable markings from a marking μ is denoted by $R(OS, \mu)$ or $R(\mu)$ if OS is clear from the context. The set of reachable markings of OS, denoted by R(OS), is the set of markings reachable from the initial marking μ_0, i.e. $R(OS) = R(OS, \mu_0)$.

The reachability graph RG(OS) is obtained as before for p/t net systems, i.e. RG(OS) is a directed graph where the set of nodes is the set of reachable markings and the (labelled) edges are the tuples $(\mu, \Theta, \mu') \in \mathcal{M} \times \Theta \times \mathcal{M}$ where $\mu \xrightarrow{\Theta} \mu'$.

We omit the mode and the EoS in the notations above if they are not relevant or clear from the context. We also say that $\widehat{t}[\vartheta]$ is enabled in μ or simply active, if a mode (λ, ρ) exists such that $\widehat{t}[\vartheta]$ is enabled in μ for (λ, ρ). This again is extended to sequences as above.

Example 3.2. To illustrate the firing rule, we return to the example of Figure 3.4. Note that the current marking μ enables $\widehat{t}[t, t']$ in the mode (λ, ρ),

where

$$\mu = \widehat{p}_1[0] + \widehat{p}_1[a+b] + \widehat{p}_2[a] + \widehat{p}_3[a'+b'] = \widehat{p}_1[0] + \lambda$$
$$\lambda = \widehat{p}_1[a+b] + \widehat{p}_2[a] + \widehat{p}_3[a'+b']$$
$$\rho = \widehat{p}_4[a+2\cdot b] + \widehat{p}_5[0] + \widehat{p}_6[c']$$

The net-tokens markings are added by the projections Π^2_N and $\Pi^2_{N'}$ resulting in the markings $\Pi^2_N(\lambda)$ and $\Pi^2_{N'}(\lambda)$. Firing the object nets transitions generates the (sub-)markings $\Pi^2_N(\rho)$ and $\Pi^2_{N'}(\rho)$. This is illustrated above and below transition $\widehat{t}$ in Figure 3.4on page 46, where the left net on top is $\Pi^2_N(\lambda)$ and the right net on top is $\Pi^2_N(\rho)$. Similar for the nets below $\widehat{t}$ for the object net N'. After the synchronisation we obtain the successor marking μ' with new net-tokens on $\widehat{p}_4, \widehat{p}_5$, and $\widehat{p}_6$:

$$\begin{aligned}\mu' &= (\mu - \lambda) + \rho = \widehat{p}_1[0] + \rho \\ &= \widehat{p}_1[0] + \widehat{p}_4[a+2\cdot b] + \widehat{p}_5[0] + \widehat{p}_5[c']\end{aligned}$$

The result is shown in Figure 3.5 on page 47

The firing rule uses a so called distributed token semantics in which the tokens of an object net may be distributed if copies of that object net are created during firing. Other semantics are possible, for examples, a value semantic where exact copies of an object net, including its internal marking, and reference semantics where a token in the system net place acts as reference to individual instances of object net. (See Valk (2003)).

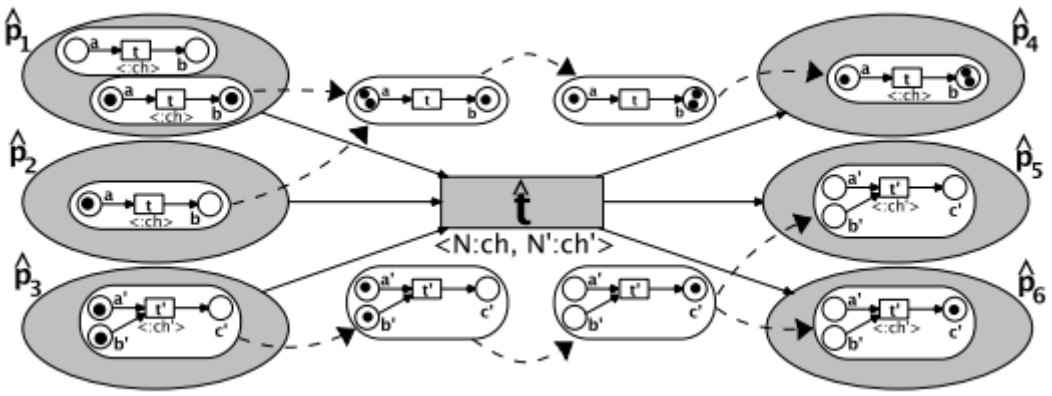

Figure 3.4.: An EOS illustrating firing rule of synchronous event $\widehat{t}[t, t']$.

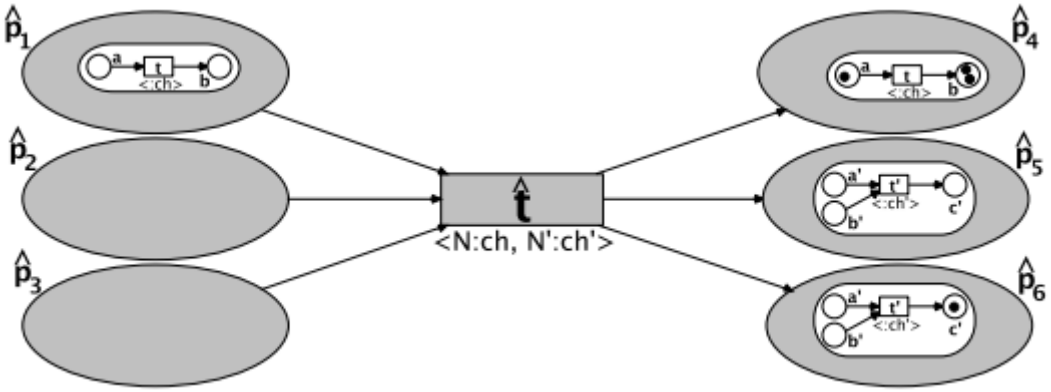

Figure 3.5.: The EOS from Figure 3.4 above after firing the synchronous event $\widehat{t}[t, t']$.

Next we now define standard decision problems for EOS analogous to problems for Petri nets.

Definition 3.7 (**Decision problems for EOS**). *1. In the reachability problem for EOS an EOS* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *and a marking* μ *of* OS *are given and the question is if* $\mu_0 \xrightarrow{*} \mu$ *holds.*

2. *In the liveness problem for EOS an EOS* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *is given and the question is if the EOS is live, i.e. if all events* $\theta \in \Theta$ *are live. An event* Θ *is live iff for all markings* μ *reachable from* μ_0 *there exists a marking* μ_0 *reachable from* μ *that enables* Θ*.*

3. *In the group liveness problem for EOS an EOS* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *and a system net transition* $\widehat{t} \in \widehat{T}$ *are given and the question is if* $\widehat{t}$ *is grouplive, i.e. if for all markings* μ *reachable from* μ_0 *there exists a marking* μ_0 *reachable from* μ *that enables* $\widehat{t}[\vartheta]$ *for some* ϑ*.*

4. *In the coverability problem for EOS an EOS* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *and a marking* μ *of* OS *are given and the question is if a marking* μ_0 *exists such that* $\mu_0 \in RS_{OS}(\mu_0)$ *and* $\mu_0 \geq \mu$ *hold.*

5. *In the boundedness problem for EOS an EOS* $OS = (\widehat{N}, \mathcal{N}, d, l, \mu_0)$ *is given and the question is if* OS *is bounded, i.e. if the set of reachable markings* $RS_{OS}(\mu_0)$ *is finite.*

For The general EOS as defined in Definition 3.1 these problems turn out to be undecidable due to highly expressive nature of the formalism. Thus require modifications and redefinition and this is the subject of the next chapter.

Chapter 4.

Elementary Reference-net Systems

4.1. Informal Introduction and Motivation

The basic elementary object systems come with some constraints in its definition that limit their expressiveness for modeling and automatic verification purposes. For instance, arc inscriptions are not associated to an arcs of a safe-EOS model making it impossible to know exactly which net tokens to remove from the input places of a system net transition and which ones to add to its output places when it fires. Also the firing rule uses a so called distributed token semantics in which the tokens of an object net may be distributed if copies of that object net are created during firing (see Definition 3.6). This firing rule incorporates the possibility to test if an object net token is the empty marking which resulted in making EOS to be a Turing-complete formalism. In this chapter, a modification to the definitions of the basic elementary object net which addresses these limitations is defined under the name elementary reference-net systems, ERS for short. The main part of the work presented here has been published in Abdullahi and Müller (2016), but this work contains some results for the computational complexity associated with translating ERS into 1-safe P/T-net.

As has already been mentioned when the basic model of EOSs has been introduced, the first difference between an EOS and a Petri net is that tokens in the system net can be nets and distinguishable. Indeed, there is a distinction between object nets and simple tokens: simple tokens are very similar to black tokens in Petri nets, while object nets have an internal state represented by a Petri net. Object nets as tokens can either change their state by means of internal autonomous transitions, or by means of interactions with other system net transitions. These interactions can be enabled or disabled depending on the internal state of the object nets located in places of the system net.

In Petri nets there is no distinction between tokens because a state is represented by a function which maps the number of tokens to each place. Therefore, each token is undistinguishable to others, but in EOS this is not necessarily true because of the structure and the internal state of object nets. Thus, the way tokens are manipulated in EOS must be different. However, since in EOS system net tokens are structured and have their own internal state, firing a system-autonomous or synchronous transition that takes a token from a place p putting it in another place could produce different results if there are several tokens in p (for example, place $\widehat{p}_1$ in Figure 3.4 on page 46 above): moving a token instead of another one could change the future behaviour of the entire EOS because certain future interactions could be enabled or may not, depending on the internal state of the moved token. A mechanism to select which tokens should be moved when a system-autonomous or synchronous transition is fired is paramount and required.

Another issue that must be taken into account is in which place a token must be placed after the execution of a transition that has more than one output place. Indeed, the way tokens are moved from input places to output places is also important and could produce different results. Looking at Figure 3.4 on page 46 it is possible to notice that again the usual Petri nets firing rule is not sufficient because it does not contain information about object net identity. For example, the object nets in the place $\widehat{p}_3$ could be moved to place $\widehat{p}_5$ or $\widehat{p}_6$ (Figure 3.5), or based on distributed token semantics, could be separated, and moved the one with empty marking to $\widehat{p}_5$ and the other one that has a marking to $\widehat{p}_6$ as shown in Figure 3.5, which may not be the intended notion.

To overcome the arc inscription constraint, firstly, we provide each marked object net located in places of the system net with unique names so that object nets with the same marking can be distinguished. That is, instead of considering marked object nets as tokens residing in places of the system net, we consider their unique names which we denote as triples (n, N, m), where n is a unique name of an object net, N is a structure of the object net from the set $\mathcal{N} = \{N_1, \ldots, N_k\}$ of object nets, and m is a marking in N. This approach is very common: (Rosa-Velardo and De Frutos-Escrig (2007)) extend P/T nets with pure name creation and name management. Secondly, we defined variables labelling arcs of the system net taken from a finite set of varibles that are bound to object nets names when firing system net transitions instead of statically typing of the system net places. This is because variables as arc inscriptions that would be bound to net names allow dynamic use of net-tokens without statically fixing type for places of the system net. A variable $v \in Var$ on arc, is interpreted either as reference to a marked object net (net-token) or the net token $N_\bullet$, which has no places or transitions so that we can also have ordinary black tokens in our set of nets.

In the definition of an ERS object nets marking are not allow to be the empty marking,(this is because nets with this capability are Turing-complete, as shown by using simulation of counter programs in Hack (1976)); again, every transition in the system net and the object net must have exactly one input place. These restrictions are imposed in other not to render the formalism practically intractable with verification in mind.

To describe the behaviour of ERS we use the *reference semantics*. In reference semantics a token in the system net place acts as reference to individual instances of object nets. The same reference can be used as a token for more than one place. Meaning that reference semantics assumes a global name space for object nets and thus consider their local marking within the system-net.

We further define some structural restrictions to ensure that new object nets can neither be created nor an existing one destroyed after a transition firing in the system net. Again, we define dynamic restriction by extending the notion of 1-safe P/T nets to ERS to guarantee that the state space is finite and show that the state space of ERS is bounded.

Moreover, we propose a set of transformation rules from 1-safe ERS into ordinary Petri nets and show isomorphism of state spaces of ERS and the generated Petri nets. Finally, we established the computational complexity for transforming 1-safe ERS into 1-safe P/T net

4.2. Formal Definition of ERS

As informally introduced in the previous section, an elementary reference-net system comprised of a system net, a set of object nets each with a unique name, variables labelling the arcs of the net system and a labelling of the transitions with channels. In order to distinguish between system and net-tokens the components of the system net will carry a hat: $\widehat{P}$, $\widehat{T}$, $\widehat{p}$, $\widehat{t}$,... etc.

4.2.1. Static Structure

Definition 4.1 (**Static Structure**). *Let the triple $\eta_i = (i, N_i, m_i)$ be a named marked object net, where i, is a unique name of an object net; N_i is a structure of the object net,*

and m_i is a marking in N_i. (Let $\Sigma = \{(i_1, N_1, m_1), \ldots, (i_k, N_k, m_k)\}$ be a finite set of unique marked named object nets). The structure of an object net with a unique name $i \in \Sigma$ is a P/T net $N_i = (P_i, T_i, F_i)$, where P_i, is the set of places of the object net, T_i is the set of its transitions and $F_i \subseteq (P_i \times T_i) \cup (T_i \times P_i)$ is the flow relation. Moreover, we assume that all sets of nodes (places and transitions) are pairwise disjoint and set $P_\Sigma = \bigcup_{\eta_i \in \Sigma} P_{\eta_i}$ and $T_\Sigma = \bigcup_{\eta_i \in \Sigma} T_{\eta_i}$. By $N_\bullet$ we denote the name of the object net which has no places or transitions denoting ordinary black tokens in our nets.

Definition 4.2 (**Elementary Reference-net System (ERS)**). *Let Var be a finite set of named variables. An elementary reference-net system is a tuple $RS = (\widehat{N}, \Sigma_{m^0}, \ell, \omega, \boldsymbol{R}^0)$ where*

- *$\widehat{N} = (\widehat{P}, \widehat{T}, \widehat{F})$ is a p/t net called a system net, where $\widehat{P}$ is its set of places, and $\widehat{T}$ is its set of transitions and $\widehat{F} \subseteq (\widehat{P} \times \widehat{T}) \cup (\widehat{T} \times \widehat{P})$ is the flow relation.*

- *$\Sigma_{m^0}\{(i_1, N_1, m_1^0), \ldots, (i_k, N_k, m_k^0)\}$,with $k \in \mathbb{N}$ is a finite set of marked named object nets.*

- *$\ell \subseteq (\widehat{T} \cup \{\widehat{\tau}\}) \times (T_{1i} \cup \{\tau\}) \times, \ldots, (T_k \cup \{\tau\}) \setminus (\widehat{\tau}, \tau, \ldots, \tau)$, is synchronisation relation, where $\widehat{\tau}$ and τ are special symbols intended to denote inactions at the system and the object net levels respectively. If $\boldsymbol{t} = (\widehat{t}, t_1, \ldots, t_k)$ and $\widehat{t}, \neq \tau$ and $\exists\, i \in \{1, \ldots, k\}$ such that $t_i \neq \tau$, then we say that $\widehat{N}$ and $N_i \in \Sigma$ for every $i \in \{1, \ldots, k\}$ with $k = |\Sigma|$, participate in $\boldsymbol{t}$. This is the reason why $(\widehat{\tau}, \tau, \ldots, \tau)$ is excluded from the set of synchronisation relation: at least one object net must participate in every synchronisation action with the system net.*

- *$\omega : \widehat{F} \longrightarrow Var \cup \{N\}$ is an arc labelling function such that for an arc $\widehat{a} \in \widehat{F}$ adjacent to a place $\widehat{p}$ the inscription of $\omega(\widehat{a})$ is a variable that will be bound to a net name.*

- *$\boldsymbol{R}^0$ specifies the initial making, where*
  $\boldsymbol{R}^0 : \widehat{P} \longrightarrow \mathbb{N} \cup MS(\Sigma)$ with $\Sigma = \{(i_1, N_1, m_1), \ldots, (i_k, N_k, m_k)\}$. It has to satisfy the condition $\boldsymbol{R}^0(\widehat{p}) \in \mathbb{N} \Leftrightarrow \boldsymbol{R}^0(\widehat{p}) \in \{N_\bullet\}$.

In the example of Figure 4.1 on page 52 an $RS = (\widehat{N}, \Sigma, \ell, \omega, \boldsymbol{R}^0)$ is shown, where $\Sigma = \{N_1, N_2\}$. Arcs of $\widehat{N}$ can be identified by their labelling from $\omega(\widehat{t})$. Hence x, y, can be bound to marked named object nets in places $\widehat{p}_1$ and $\widehat{p}_2$ adjacent to transition $\widehat{t}$ to enable it. In the initial marking, places $\widehat{p}_1$ and $\widehat{p}_2$ contain references to the marked named object nets N_1 and N_2 respectively. They have the same structure and could be generated from a

type N. $\mathcal{N} = \{i|(i, N_{i,}, m_i) \in \Sigma\}$, *is a finite set of object nets names. Moreover, variables appearing on arcs adjacent to a transition* $\widehat{t}$ *of the system net must satisfy the following conditions:*

$$\forall \widehat{t} \in \widehat{T} \, and \, \forall \widehat{p} \in {}^\bullet\widehat{t}, \exists \, \widehat{p}' \in \widehat{t}^\bullet, \, such \; that \; \omega(\widehat{p}, \widehat{t}) = \omega(\widehat{t}, \widehat{p}') \; or \; \omega(\widehat{p}, \widehat{t}) = N_\bullet \tag{4.1}$$

$$\forall \widehat{t} \in \widehat{T} \, and \, \forall \widehat{p} \in {}^\bullet\widehat{t}, \exists \, \widehat{p}' \in \widehat{t}^\bullet, \, such \; that \; \omega(\widehat{p}', \widehat{t}) = \omega(\widehat{t}, \widehat{p}) \; or \; \omega(\widehat{p}, \widehat{t}) = N_\bullet \tag{4.2}$$

$$\forall \widehat{t} \in \widehat{T} \; and \; for \; any \; two \; places \; \widehat{p}_1, \widehat{p}_2, \in {}^\bullet\widehat{t}, \; if \; \widehat{p}_1 \neq \widehat{p}_2 \; then \; \omega(\widehat{p}_1, \widehat{t}) \neq \omega(\widehat{p}_2, \widehat{t}). \tag{4.3}$$

$$\forall \widehat{t} \in \widehat{T} \; and \; for \; any \; two \; places \; \widehat{p}'_1, \widehat{p}'_2, \in \widehat{t}^\bullet, \; if \; \widehat{p}_1 \neq \widehat{p}_2 \; then \; \omega(\widehat{t}, \widehat{p}'_1) \neq \omega(\widehat{t}, \widehat{p}_2) \, . \tag{4.4}$$

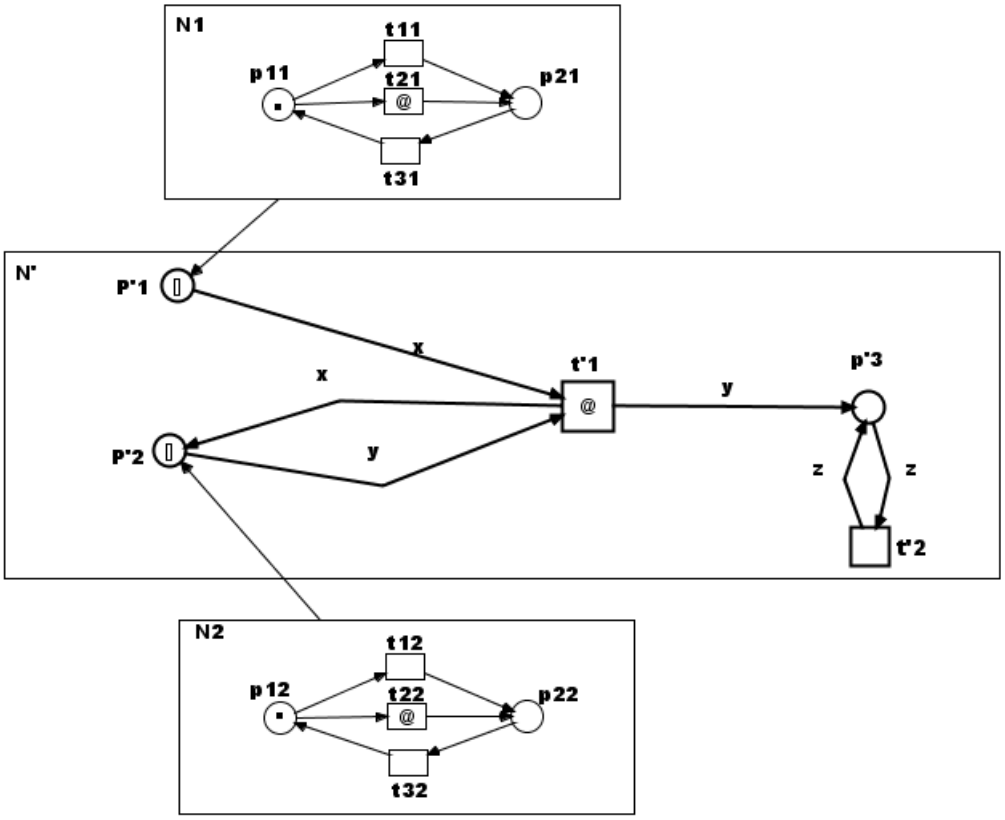

Figure 4.1.: Example of ERS

Figure 4.1 (on page 52) shows an example of an ERS satisfying these conditions. Condition (1) says that each variable appearing in the incoming arc of a system net transition $\widehat{t}$ also has to appear in an outgoing arc of $\widehat{t}$ or no such variable exist. Condition (2) says that each variable appearing in an outgoing arc of a system net transition $\widehat{t}$ also has to appear in an incoming arc of $\widehat{t}$ or no such variable exist. These two conditions show that dynamic change of the number of object nets is not allowed i.e., no new object net is created and no existing one is destroyed after a transition firing in the system net. This kind of restriction can be seen in other works such as the one in Frumin and Lomazova

(2014). Condition (3) prevents the ability to join two object nets, and (4) prevents the splitting of an object nets. (This is because in reality, complex physical entities cannot be cloned at run time).

It is paramount to note that with these structural restrictions, ERS still retain the capability to describe nesting of object nets, synchronisation, and mobility, but do not allow splitting of the inner marking of an object net or joining the inner marking of several object nets. For instance, if assuming these inner markings as modelling the inner state of an agent, this is a reasonable restriction and shows that ERSs are well suitable to model physical entities

4.2.2. Dynamic Behaviour

We start by introducing the notion of marking for elementary reference-net system ERS under reference semantics. Recall that by Definition 4.2 on page 51. an ERS contains a set $\Sigma_{m^0}\{(i_1, N_1, m_1^0), \ldots, (i_k, N_k, m_k^0)\}$ of marked named object nets. By ignoring the marking we obtain the set of (unmarked) named object nets $\Sigma\{(i_1, N_1), \ldots, (i_k, N_k)\}$. Hence in general a marking is given by

1. A distribution of object nets or black tokens $R : P \longrightarrow \mathbb{N} \cup MS(\Sigma)$ and

2. The vector $\mathbf{M} = (m_1, \ldots, m_k)$ with the current marking of each N_i $(1 \leq i \leq k)$.

$\mathbf{R}$ specifies for each system net place $\widehat{p}$ a number of black tokens (if $\widehat{p}$ contains a black tokens) or a multiset of unmarked named object nets (if $\widehat{p}$ contain reference(s) to unmarked named object nets). If we abbreviate $(m_1, \ldots, m_k)$ by $\mathbf{M}$ and the set of all such vectors by $\mathcal{M}$ we obtain the following Definition 4.3 below. By $\Pi_i(\mathbf{M})$ we denote the $i - th$ component m_i of $\mathbf{M}$ and by $\mathbf{M}_{i \longrightarrow m_i}$ the tuple, where the $i - th$ component is substituted by m_i.

In what follows a marked named object net is referred to as net-token. For a given ERS, by $\Sigma_{nt} = \Sigma \cup \{N_\bullet\}$ we denote the set of all marked named net-tokens.

Henceforth, instead of considering net-tokens (marked object nets residing in a place of the system net), we consider names of object net tokens $i = (i, N_i, m_i)$. Then every token in a system net place refers to a unique name $i \in \Sigma_{nt}$.

Sometimes by abuse of notation, for a named object net $i = (i, N_i, m_i)$ in a place $\widehat{p}$ of a marking $\mathbf{R}$ of the system net we write $\mathbf{R}(\widehat{p}) = i$ meaning $\mathbf{R}(\widehat{p}) = \{(i, N_i, m_i))\}$.

Definition 4.3. *Given an elementary reference-net system* $RS = (\widehat{N}, \Sigma_{nt}, \ell, \omega, \boldsymbol{R}^0)$ *we define* $\mathcal{M} = \{M \mid M = (m_1, \ldots, m_k) \wedge m_i \in MS(P_i)\}$. *Then a marking of an elementary reference-net system is a pair* (R, M) *where* $M \in \mathcal{M}$ *and* $\boldsymbol{R} : \widehat{P} \longrightarrow MS(\Sigma_{nt})$ *satisfying the condition* $\boldsymbol{R}^0(\widehat{p}) \in \mathbb{N} \Leftrightarrow \boldsymbol{R}^0(\widehat{p}) \in \{N_\bullet\}$. *Specifying* M^0 *by the initial markings of the marked named object nets* $M^0 = (m_1^0, \ldots, m_k^0)$ *we obtain the initial marking* $(\boldsymbol{R}^0, \boldsymbol{M}^0)$ *of* RS.

The set of all markings of RS *is denoted by* $\mathcal{M}_r$.

Like in other classes of High-level nets, variables are bound to names from an adjacent place $\widehat{p}$ of a marking $\mathbf{R}$ of the system net when firing a transition in order to determine which tokens are removed from pre-conditions and which are added to post-conditions.

Let $\widehat{t} \in \widehat{T}$ be a transition in the system net $\widehat{N}$, then $\bullet\widehat{t} = \{\widehat{p} \mid (\widehat{p}, \widehat{t}) \in \widehat{F}\}$, and $\widehat{t}\bullet = \{\widehat{p} \mid (\widehat{t}, \widehat{p}) \in \widehat{F}\}$ are sets of its pre- and post-conditions. We denote by $\omega(\widehat{t}) = \{\omega(\widehat{p}, \widehat{t}) \mid (\widehat{p}, \widehat{t}) \in \widehat{F}\} \cup \{\omega(\widehat{t}, \widehat{p}) \mid (\widehat{t}, \widehat{p}) \in \widehat{F}\} = \bullet\widehat{t} \times \{\widehat{t}\} \cup t \times \widehat{t}\bullet$ the set of all variables on arcs adjacent to $\widehat{t}$.

A binding function β specifies which variables are bond to names, where $\beta \colon \omega(\widehat{t}) \cup \{N_\bullet\} \longrightarrow \mathcal{N} \cup \{\bullet\}$ with $\mathcal{N} = \{i \mid (i, N_i, m_i) \in \Sigma\}$. Satisfying the conditions: for each $x \in \omega(\widehat{t}) \cup \{N_\bullet\}$, there exist $i \in N$ such that $\beta(x) = i$ and if $x = \bullet$ then $\beta(x) = N_\bullet$.

The *firing rule* will be introduced in three modes similar to those of EOS. First we consider the mode when synchronisation occurs. In this mode we assume that a system net transition $\widehat{t} \in \widehat{T}$ and one or more object nets transition $t_i \in T_\Sigma$ of some object nets $N_i (i \geq 1)$ are activated and all transitions are related by the synchronisation relation ℓ. That is, $(\widehat{t}, t_i) \in \ell$. This mode of the firing rule is called a synchronisation firing mode.

Definition 4.4 ((synchronous firing mode). *Let* $(\boldsymbol{R}, \boldsymbol{M})$ *be a marking of an elementary reference-net system* $RS = (\widehat{N}, \Sigma_{nt}, \ell, \omega, \boldsymbol{R}^0)$, $\widehat{t} \in \widehat{T}$ *a transition of* $\widehat{N}$ *and let* β *be a variable binding function defined for all* $x \in \omega(\widehat{t}) \cup \{N_\bullet\}$. *Let* $\alpha_1, \ldots, \alpha_k \in \Sigma_{nt}$ *be object nets involved in the firing of* $\widehat{t}$. *Then* $\widehat{t}$ *can fire provided that in each* $\alpha_i \in \Sigma_{nt}$ *for every* $i \in (1, \ldots, k)$ *there is a transition* $t_i \in T_\Sigma$ *such that* $(\widehat{t}, t_1, \ldots, t_k) \in \ell$. *Then* $(\widehat{t}, t_1, \ldots, t_k)$ *is activated in* $(\boldsymbol{R}, \boldsymbol{M})$ *if:*

$$\forall \widehat{p} \in \widehat{P}, (\beta(\omega(\widehat{p},\widehat{t})), N_{\beta(\omega(\widehat{p},\widehat{t}))}, m_{\beta(\omega(\widehat{p},\widehat{t}))}) \in \boldsymbol{R}(\widehat{p}) \, and \\ \Pi_i(\boldsymbol{M}) \geq 1 \, if, (p_i, t_i) \in F_i \forall p_i \in P_i. \quad (4.5)$$

This is denoted by $(R, M)[\widehat{t}, (t_1 \dots t_k)\rangle$ *Let be* $m_i[t_i\rangle m_i'$ *(w.r.t* N_i*) In this mode the successor marking* $(\boldsymbol{R}', \boldsymbol{M}')$ *is defined by*

$$\boldsymbol{R}'(\widehat{p}) = R(\widehat{p}) \setminus (\beta(\omega(\widehat{p},\widehat{t})), N_{\beta(\omega(\widehat{p},\widehat{t}))}, m_{\beta(\omega(\widehat{p},\widehat{t}))}) \cup \\ (\beta(\omega(\widehat{t},\widehat{p})), N_{\beta(\omega(\widehat{t},\widehat{p}))}, m_{\beta(\omega(\widehat{t},\widehat{p}))}) \, \forall \widehat{p} \in \widehat{P} \; and \; \boldsymbol{M}' = \boldsymbol{M}_{i \longrightarrow m_i} \quad (4.6)$$

This is denoted by $(\boldsymbol{R}, \boldsymbol{M})[\widehat{t}, t_i\rangle(\boldsymbol{R}', \boldsymbol{M}')$.

In our running example of RS from Figure 4.1 on page 52 with $(\boldsymbol{R}, \boldsymbol{M}) = (\boldsymbol{R}^0, \boldsymbol{M}^0)$, $\ell = \{(\widehat{t}, t_1 t_2)\}$, $\widehat{t} = \widehat{t}_1$, $(\beta_1(\omega(\widehat{p}_1, \widehat{t}_1)), N_{\beta(\omega(\widehat{p}_1,\widehat{t}_1))}, m_{\beta(\omega(\widehat{p}_1,\widehat{t}_1))})$ *with* $\omega(\widehat{p}_1, \widehat{t}_1) = x$ *and* $\beta(x) = 1$ *the object net that involved in the synchronisation firing is* $N_1 = (1, N_1, m_1)$ *and* $t_1 = t_{21}$, $(\beta_2(\omega(\widehat{p}_2, \widehat{t}_2)), N_{\beta(\omega(\widehat{p}_2,\widehat{t}_2))}, m_{\beta(\omega(\widehat{p}_2,\widehat{t}_2))})$ *with* $\omega(\widehat{p}_2, \widehat{t}_2) = y$ *and* $\beta(y) = 2$ *the object net that involved in the synchronisation firing is* $N_2 = (2, N_2, m_2)$ *and* $t_2 = t_{22}$.

We obtain $(\boldsymbol{R}, \boldsymbol{M})[\widehat{t}_1, t_{21}, t_{22}\rangle(\boldsymbol{R}', \boldsymbol{M}')$ *i.e., a marking* $(\boldsymbol{R}', \boldsymbol{M}')$ *is reached where places* $\widehat{p}_2$ *and* p_3 *contain references to net-tokens, namely* $R'(\widehat{p}_2) = 1$ *and* $R'(\widehat{p}_3) = 2$ *and* M' *is the marking reached for each net-token that participated in the synchronisation firing mode with* $\boldsymbol{M}_{1 \longrightarrow m_1} = m_1 = \{p_{21}\}$, *and* $\boldsymbol{M}_{2 \longrightarrow m_2} = m_2 = \{p_{22}\}$.

If a system net transition is activated and without being included in the synchronisation relation, a chosen net-token does not change it current marking. As it changes its location in the system net such a firing mode is called system-autonomous. The following definition can be seen as a special case of Definition 4.4 where the involved net-tokens markings are not changed, i.e. $\mathbf{M}' = \mathbf{M}$.

Definition 4.5 ((System-autonomous firing mode). *Let* $(\boldsymbol{R}, \boldsymbol{M})$ *be a marking of an elementary reference-net system* $RS = (\widehat{N}, \Sigma_{nt}, \ell, \omega, \boldsymbol{R}^0)$ *and* $\widehat{t} \in \widehat{T}$ *a transition of* $\widehat{N}$ *with a binding* β *such that* $\widehat{t} \notin dom(\ell) = \{\widehat{t}_1 \mid \exists \widehat{t}_i : (\widehat{t}_1, t_i) \in \ell . \forall i \geq 1\}$. *Then* $\widehat{t}$ *is activated in* $(\boldsymbol{R}, \boldsymbol{M})$ *if there is a net token such that*

$$(\beta(\omega(\widehat{p},\widehat{t})), N_{\beta(\omega(\widehat{p},\widehat{t}))}, m_{\beta(\omega(\widehat{p},\widehat{t}))}) \in \boldsymbol{R}(\widehat{p}) \, \forall \widehat{p} \in \widehat{P} \quad (4.7)$$

Since we use τ, for inaction, this is denoted by $(\boldsymbol{R},\boldsymbol{M})[\widehat{t},\tau\rangle$. In this mode the successor marking $(\boldsymbol{R}',\boldsymbol{M}')$ is defined by

$$\forall \widehat{p} \in \widehat{P} : \boldsymbol{R}'(\widehat{p}) = \boldsymbol{R}(p) \setminus (\beta(\omega(\widehat{p},\widehat{t})), N_{\beta(\omega(\widehat{p},\widehat{t}))}, m_{\beta(\omega(\widehat{p},\widehat{t}))}) \cup ((\beta(\omega(\widehat{t},\widehat{p})), N_{\beta(\omega(\widehat{t},\widehat{p}))}, m_{\beta(\omega(\widehat{t},\widehat{p}))}) \, and \, \boldsymbol{M}' = \boldsymbol{M} \quad (4.8)$$

This is denoted by $(\boldsymbol{R}.\boldsymbol{M})[\widehat{t}_1,\tau\rangle(\boldsymbol{R}',\boldsymbol{M}')$.

Definition 4.6 ((object autonomous firing mode). *Let $(\boldsymbol{R},\boldsymbol{M})$ be a marking of an elementary reference-net system $RS = (\widehat{N},\Sigma_{nt},\ell,\omega,\boldsymbol{R}^0)$ and $t_i \in T_i$ a transition of a net-token $i = (i,N_i,m_i) \in \boldsymbol{R}(\widehat{p})$ for some $\widehat{p} \in \widehat{P}$, such that $\nexists(\widehat{t},x_i,\ldots,t_i,\ldots,x_k) \in \ell$, and t_i iss activated in N_i. Then we say that $(\widehat{\tau},t_i$ is activated in $(\boldsymbol{R},\boldsymbol{M})$ (denoted $(R,M)[(\widehat{\tau},t_i\rangle$. The successor marking $(\boldsymbol{R}',\boldsymbol{M}')$ of RS is defined by*

$$\boldsymbol{R}' = \boldsymbol{R} \, and \, \boldsymbol{M}' = \boldsymbol{M}_{1 \longrightarrow m_i} \, if \, m_i[t_i\rangle m_i' \, for \, \Pi_i(\boldsymbol{M}) = m_i. \quad (4.9)$$

This is denoted by $(\boldsymbol{R},\boldsymbol{M})[\widehat{\tau},t_i\rangle(\boldsymbol{R}',\boldsymbol{M}')$.

Definitions 4.4, and 4.6 could be easily merged. However, this is not done here to emphasise the differences.

To introduce the occurrence sequences for *ERS* we assume an *ERS* as defined in Definition 4.2 on page 51. Let *RS* be an *ERS* and $(\mathbf{R},\mathbf{M}),(\mathbf{R}',\mathbf{M}') \in \mathcal{M}_r$.

Definition 4.7. *For a new alphabet $\Gamma = (\widehat{T} \cup \{\widehat{\tau}\}) \times (T_1 \cup \{\tau\})\times,\ldots,(T_k \cup \{\tau\}) \setminus (\widehat{\tau},\tau,\ldots,\tau)$ where $(\widehat{\tau},\tau,\ldots,\tau)$ denotes the neutral element of Γ^*, we define:*

$$\begin{aligned}(\boldsymbol{R},\boldsymbol{M})[(\widehat{\tau},\tau,\ldots,\tau)\rangle(\boldsymbol{R}',\boldsymbol{M}') \, iff \, (\boldsymbol{R},\boldsymbol{M}) = (\boldsymbol{R}',\boldsymbol{M}') \, and \\ (\boldsymbol{R},\boldsymbol{M})[\breve{w}(\widehat{t},\alpha)\rangle(\boldsymbol{R}',\boldsymbol{M}') \, if \, \exists(\boldsymbol{R}'',\boldsymbol{M}'') : (\boldsymbol{R},\boldsymbol{M})[\breve{w}\rangle(\boldsymbol{R}'',\boldsymbol{M}'') \, and \\ (\boldsymbol{R}'',\boldsymbol{M}'')[(\widehat{t},\alpha)\rangle(\boldsymbol{R}',\boldsymbol{M}') \, for \, some \, \breve{w} \in \Gamma^*, \widehat{t} \in \widehat{T} \cup \{\widehat{\tau}\} \, and \, \alpha \in ((T_1 \cup \{\tau\})\times,\ldots,(T_k \cup \{\tau\})).\end{aligned} \quad (4.10)$$

To denote that $(\boldsymbol{R}',\boldsymbol{M}')$ is reachable from $(\boldsymbol{R},\boldsymbol{M})$ by some occurrence sequence of actions we write $(\boldsymbol{R},\boldsymbol{M}) \xrightarrow{*} (\boldsymbol{R}',\boldsymbol{M}')$.

The set of reachable markings of a reference system RS from a marking $(\boldsymbol{R},\boldsymbol{M})$ is denoted

by $R(RS,(\boldsymbol{R},\boldsymbol{M}))$. $R(RS)$, is the set of markings reachable from the initial marking $(\boldsymbol{R}^0,\boldsymbol{M}^0)$, i.e. $(R(RS)=R(RS,(\boldsymbol{R}^0,\boldsymbol{M}^0))$. The reachability graph $(RG(RS)$ is obtained as for P/T-net systems, i.e. $RG(RS)$ is a labelled directed graph whose nodes is the set of reachable markings and edges are the tuples $((\boldsymbol{R},\boldsymbol{M}),(\widehat{t},\alpha),(\boldsymbol{R}',\boldsymbol{M}'))\in\mathcal{M}_r\times(\widehat{t},\alpha)\times\mathcal{M}_r$ where $(\boldsymbol{R},\boldsymbol{M})\xrightarrow{(\widehat{t},\alpha)}(\boldsymbol{R}',\boldsymbol{M}')$.

We now extend the definition of 1-safe P/T-net to ERS. Safeness guarantees that the state space of a P/T-net is finite and for a 1-safe net, at any point in time in each place there is at most one token. Many problems for 1-safe nets e.g., reachability, and liveness become decidable in polynomial space (Esparza, 1996). A P/T net is 1-safe if and only if for all reachable marking there is at most one token on each place. In a 1-safe P/T-net all reachable markings can be interpreted as a set (of marked places)

We introduce two conditions for safeness of ERS in Definition 4.8 as a generalisation of the safeness notion for P/T-nets

Definition 4.8 ((1-safe ERS). *Let $RS=(\widehat{N},\Sigma,\ell,\omega,\boldsymbol{R}^0)$ be an ERS. RS is 1-safe if and only if all reachable markings are 1-safe and if and only if in all reachable markings there is at most one net-token on each system net place and each net-token is 1-safe:*

- $\forall(\boldsymbol{R},\boldsymbol{M})\in R(RS),\forall\widehat{p}\in\widehat{P}:(R(\widehat{p})\leq 1)$ *and*
- $\forall(i,N_i,m_i)\in\boldsymbol{R}(\widehat{p}):\forall p_i\in P_i\forall\widehat{p}\in\widehat{P}(\boldsymbol{R}(\widehat{p}),\Pi_i(\boldsymbol{M}(p_i))>0\Rightarrow\Pi_i(\boldsymbol{M}(p_i))\leq 1.$

Note that by this definition in the reachable marking of safe ERS each system net place is marked with at most one net-token. The set $\{\mathbf{R}(\widehat{p})\mid\mathbf{R}(\widehat{p})\in R(RS),\widehat{p}\in\widehat{P}\}$ is thus a finite set and similar to the reachability set of safe P/T-net. Furthermore, the net-tokens are also safe by the definition and thus the set of reachable markings associated with them is finite too. This combination of the finite set of reachable markings associated with the system net and finite set of reachable markings associated the net-tokens results in a finite state space for the ERS.

Observation 4.1. *Given an ERS if for all reachable markings there is at most one token on each system net place and each net-token is 1-safe, then all reachable markings are 1-safe.*

Theorem 4.1. *If an ERS is safe, then its set of reachable markings is finite.*

Proof. By definition of safe ERS each net token is 1-safe and hence there are at most 2^n different markings a net-token may have (with n the number of places). Again, by

definition of safe ERS each system net place is either marked or unmarked with a net-token with one of these markings, thus there are up to $(1 + 2^n)^m$ different markings of RS, i.e. $|R(RS)| \leq (1 + 2^n)^m$ and m the number of net-tokens. □

4.3. Transformation of Elementary Reference-net System into P/T- nets

4.3.1. Basic concept on Transforming ERS into P/T-nets

We will construct a finite P/T-net model for the entire ERS model which has behavioural equivalence by establishing a strong bisimulation equivalence between states of the two models. By doing so, we can develop a set of transformation rules that allow us to apply target model having the same behavioural properties like the original one for formal verification and analysis.

Related works have been reported in Lomazova and Ermakova (2016). Hence, we highlight here the similarities and differences between the proposed approach and this related study. The approach are closer to our work since they adapt the nets-within-nets paradigm that allow tokens to be nets themselves. Another similarity is the purpose of the studies, which main goal is to establish methods for formal verification of a class of *Nets-within-Net* formalisms.

The major similarities between our work and that of Lomazova and Ermakova (2016) is that they developed a set of rules for translating a safe conservative nested Petri nets (NP-net) into an equivalent P/T net. Some main differences are that we established clearly, an important relation between the isomorphic properties of state spaces of safe-ERS and 1-safe P/T net. Among such results are the established Lemmas, and proof of a theorem for the isomorphism. Moreover, we adopt a different way of introducing the procedure for transforming Nets-within-Nets into 1-safe P/T net, which consequently, gives a neater and easier-to-understand presentation. In particular we used not only constructive proofs but also, non-constructive proofs (see Section 4.4). Finally, we established the computational complexity for transforming safe-ERS into 1-safe P/T net.

In ERS every token in the system net refers to a unique name taken from the set of all marked named net-tokens. During firing a synchronous transition, a token is passed from

the input places of a synchronous transition to its output places, also for each net-token involved in the synchronisation firing, black tokens are removed from the input places of its transition and brought to its output places. This can be realised by transitions relation sets. The transition relation, however, cannot be used in P/T-nets reachability analysis. The major reasons for this is that tokens in an ERS may have a reference of a net-token, however, tokens in P/T-nets do not hold such kind of information. To overcome this problem, we will duplicate transitions together with their appropriate input and output arcs in our P/T-net for a family of synchronisation transitions which belongs to the system net, and transitions which belong to the subset of the transition relation set in each marked named net-token residing in the initial marking of an ERS based on the concept in Lomazova and Ermakova (2016) which translate a Nested Petri Nets into behaviourally equivalent P/T-net.

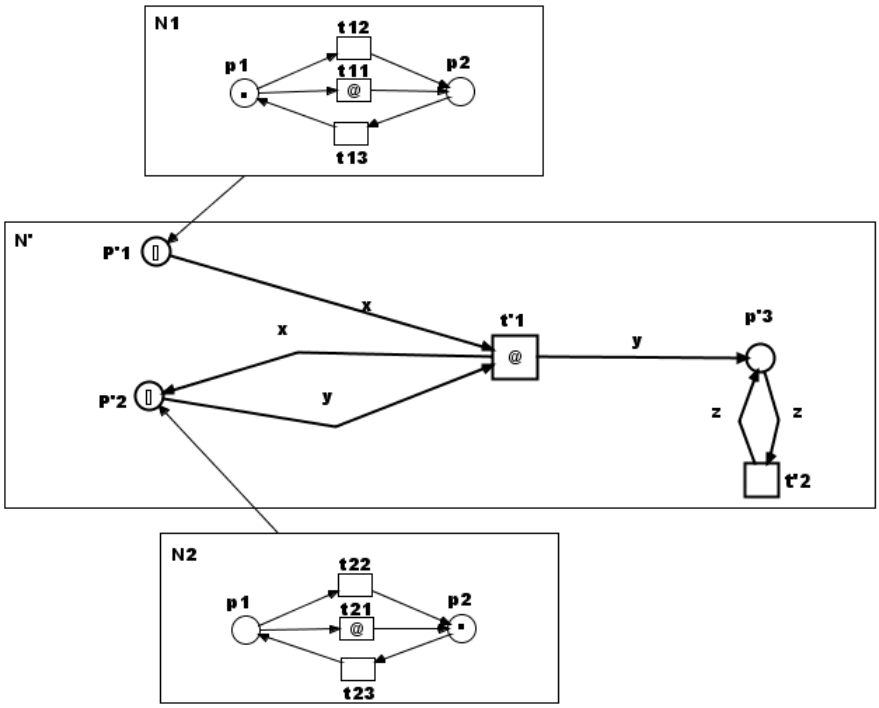

Figure 4.2.: A model of Elementary Reference System

4.3.2. Transformation Rules

The main idea is to construct a bi-simulating P/T-net where the set of places is identified by tuples $(\widehat{p}, i)$ where $\widehat{p}$ is a system net place and i ranges over all marked object net names that reside on $\widehat{p}$. This set is finite due to the boundedness of the ERS. The set of transitions coincides with the set of transition relation $(\widehat{t}, t_1, \ldots, t_k) \in \ell$ in the marking

$(\mathbf{R}, \mathbf{M})$ that is encoded in the connections from the places to these transitions. Also since ERS is finite there are only finitely many firing modes and hence the set of transition is also finite.

In the following, we present a set of transformation rules for Elementary Reference-net system (Section 4.2.). There are five rules and they must be applied in sequence from Rule 1 to Rule 5. The first rule generates the set of places of the target P/T-net. The second rule defines the initial marking for the P/T-net. The third rule generates a family of transitions and arcs for each autonomous transition in the system net, and rule 4 generates family of transitions and arcs for each autonomous transition in a net-token from the set off all marked named net-tokens. The fifth rule creates a family of synchronisation transitions which belongs to the system net, and synchronization transitions in each net-tokens by combining Rule 3 and Rule 4.

The set of transformation rules will be illustrated with the example of an elementary reference-net system shown in Figure 4.2 on page 59. In the figure, there is the system net N represented by a net in the middle. Tokens residing in places p_1 and p_2 are references to marked named net-tokens N_1 and N_2. Their structures and inner markings are shown on the top and bottom of the system net. This net system will be translated into a P/T net system N^*.

Let $RS = (\widehat{N}, \Sigma, \ell, \omega, \mathbf{R}^0)$ be an *ERS* with a set Σ_{nt} of all marked named net tokens in the initial marking. By R we denote the set of all names used in Σ_{nt} and by $R_i \subseteq R$ the subset of all names for marked net token with name i. The net system RS will be translated into a flat P/T-net system $N^* = (P^*_N, T^*_N, F^*_N, M^*_0)$ where M^*_0 is the initial marking.

Rule 1: Generate places. To generate the set $P^*_{N^*}$ of places of a P/T-net N^* we define two separate sets. The first, is the set P'_{N^*} of places from the system net $\widehat{N}$, and the second, the set P_{N^*} of all places of each net-token in the initial marking of the system net. Finally, we take the union of these set as the set $P^*_{N^*}$ of a target P/T-net N^*, with the assumption that $P'_{N^*} \cap P_{N^*} = \phi$

The first set of places of N^* is generated by duplicating all places of the system net for each net-token name (i, N_i, m_i), $(i \geq 1)$ used in the initial marking of the system net and labelled it with a pair $(\widehat{p}', i)$ where p' is a place in $\widehat{P}$ and i is the name of the possible net-token that reside on p'. Thus the set P'_{N^*} of places of N^* from place of the system

net is defined as follows:

$$P'_{N^*} = \cup_{p' \in \widehat{P}} \{(p', i) \mid i \in R, i \geq 1\}. \tag{4.11}$$

The second set of places of N^* is generated by taking a copy of each place in the set P_i for each net-token and labelled it with a pair (p_i, i) where p_i is a place in P_i and i is the name of the net-token. Thus the set P_{N^*} of N^* from places of each net-token is defined as follows:

$$P_{N^*} = \cup_{i \in \Sigma_{nt}} \{(p_i, i) \mid p_i \in P_i, i \in R, i \geq 1\} \tag{4.12}$$

Therefore the set $P^*_{N^*}$ of a target P/T-net N^* is the union of these set, namely

$$P^*_{N^*} = P'_{N^*} \cup P_{N^*} \tag{4.13}$$

For the *ERS* in Figure 4.2 on page 59, each place in system net is duplicated and labelled with each net-token name represented in the initial marking as $(p'_1, 1)$, $(p'_1, 2)$, $(p'_2, 1)$, $(p'_2, 2)$, $(p_3, 1)$, $(p'_3, 2)$ and one copy of places in the set P_i for each net-token N_i as $(p_1, 1)$, $(p_2, 1)$, $(p_1, 2)$, and $(p_2, 2)$ in P/T-net N^* from Rule 1 as shown in Figure 4.3 on page 62.

Rule 2: Define the initial marking for N^*. For a P/T-net N^* we define an encoding of markings on places from the set of places $\widehat{P}$ in an *ERS* by markings on the generated places from $P^*_{N^*}$. If a net-token with name $i \in R_i$ resides in a place $\widehat{p}$ in an initial marking $R^0(\widehat{p})$ of the system net, then the number of black tokens in place $(\widehat{p}, i) \in P^*_{N^*}$ in the initial marking M^*_0 in the constructed net is the number of appearances of name i in the multiset $R^0(\widehat{p})$, namely

$$M^*_0(\widehat{p}, i) = R^0(\widehat{p}). \tag{4.14}$$

Also, we define an encoding of markings on places from the set of places P_i on the generated places from $P^*_{N^*}$. If all places (p, i) for all p such that $(p, i) \in P^*_{N^*}$ is marked in the initial marking M^0 of the net-token $i \in R_i$, then the number of black tokens in place $(\widehat{p}, i) \in P_{N^*})^*$ in M^*_0 is the number black tokens in $M^0(p)$, namely

$$M^*_0(p, i) = M^0(p). \tag{4.15}$$

If a place in the system net is a place that contains a black token, then the unique copy corresponding to the place in N^* is also marked with a black token.

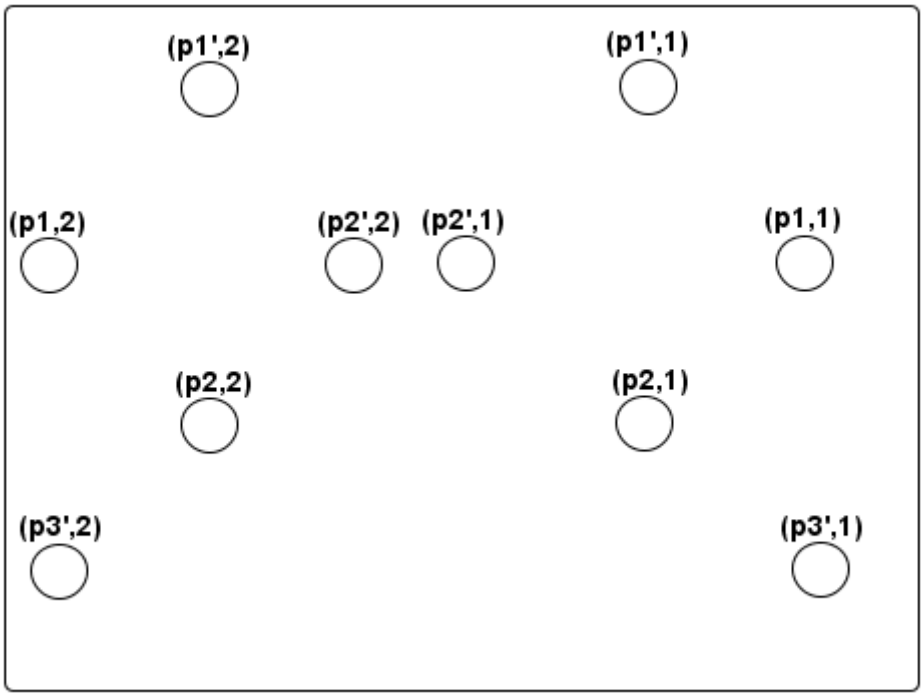

Figure 4.3.: Set of places of P/T net N^*

It is easy to see that this encoding defines a one-to-one correspondence between marking in *ERS* and 1-safe markings in N^*. In the given *ERS*, reference to the net-token N_1 resides in $\widehat{p}_1$, and reference to the net-token resides in $\widehat{p}_2$. Hence, we have tokens in $(p'_1, 1)$ and $(p'_2, 2)$ for N^*. Likewise, we define the markings for places $(p_1, 1)$ and $(p_2, 2)$. This is illustrated in Figure 4.4 on page 63 .

Rule 3: Generate a family of P/T-net transitions from a system net. We define a set T^*_{sat} of transitions of N^* obtained from each autonomous transition of the system net $\widehat{N}$ by duplicating each autonomous transition for each input arc variable of $\widehat{t}$ that may be bound to any of the named net-token name in each place adjacent to $\widehat{t}$ with appropriate input and output arcs, in N^*.

$$T^*_{sat} = \cup_{\widehat{t} \in \widehat{T}} \{ t'_{\beta_{i(x)}} \mid x \in \omega(\widehat{t}) : \widehat{t} \text{ is a system autonomous transition} \} \tag{4.16}$$

In the example *ERS*, the set $\omega(\widehat{t})$ of input arc variables that can be bound to a named

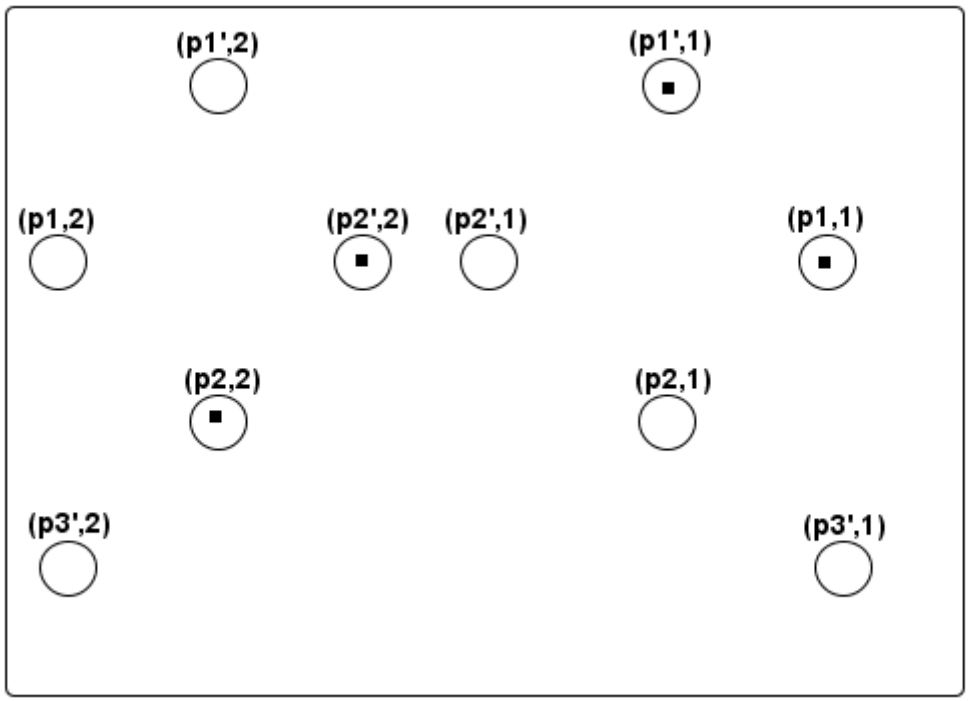

Figure 4.4.: Set of places of P/T net N^* with the initial marking

net-token for t'_2 is as follows:

$$\beta(\omega(t'_2)) = \{\beta_1 = (z = 1)\beta_2 = (z = 2)\}. \tag{4.17}$$

where in binding β_1 the named net-token 1 is bound to the input arc variable z and in binding β_2, the named net-token 2 is also bound to the input arc variable z, respectively.

Therefore, two transitions t'_{21} and t'_{22} are generated for transition t'_2 from Rule 3.

The appropriate input and output arcs are drawn so as to keep input and output places for each transition representing an autonomous transition $t' \in T^*_{sat}$ in N^*. We define a set F^*_{sat} representing arcs of system autonomous transitions in N^* as follows:

$$F^*_{sat} = \bigcup_{\widehat{a} \in \widehat{F}} \{(x', y') \mid (x, y) = \omega(\widehat{a}), x' \in P'_{N^*}(x) \cup T^*_{sat}(x), y' \in P'_{N^*}(y) \cup T^*_{sat}(y)\}. \tag{4.18}$$

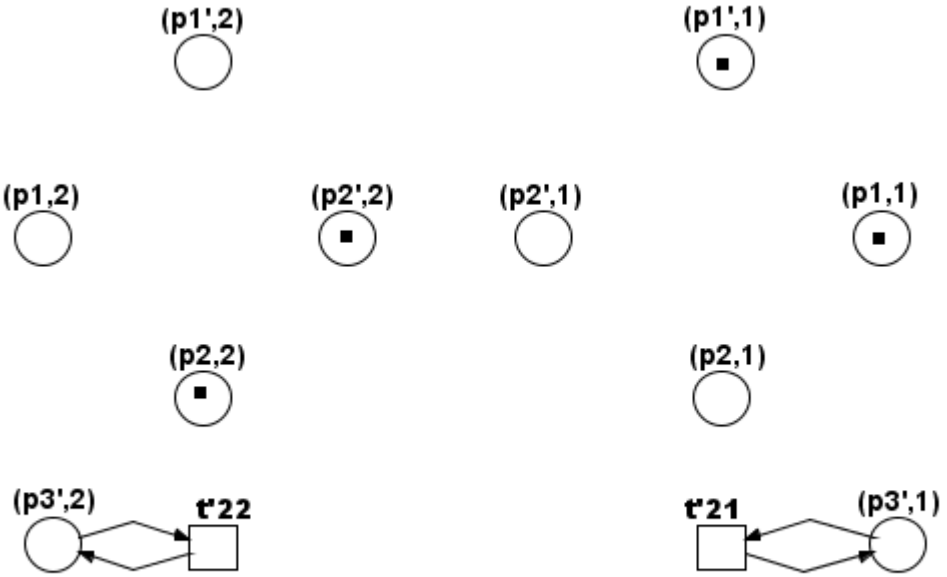

Figure 4.5.: Transitions and arcs generated after Rule 3

The set F^*_{sat} gives all possible pairs of a places, arcs and a transition representing each system autonomous transition to be drawn in N^*. This is shown in Figure 4.5.

Rule 4: Generate family of transitions representing autonomous transitions in each net-token. For a set T^*_{nat} of transitions of N^* obtained from each autonomous transition in each net-token with name $i = (i, N_i, m_i) \in \Sigma_{nt}$ we define a set of similar autonomous transitions as follows:

$$T^*_{nat} = \bigcup_{i \in \Sigma_{nt}} \{t \mid t_i \in T_i \wedge t_i \notin \ell\}. \tag{4.19}$$

In the example four transitions t_{21}, t_{22}, t_{31}, t_{32} are generated. For each autonomous transitions t_{21} and t_{31} in net-token $i = 1$, similar transitions $t_1 1$ and $t_3 1$ are generated in N^*, also, for each autonomous transitions t_{22} and t_{32} in net-token $i = 2$, similar transitions t_{22} and t_{32} are generated in N^* from Rule 4.

Again, appropriate input and output arcs are drawn so as to keep input and output places for each transition representing a net-token autonomous transition $t \in T^*_{nat}$ in N^*. Let us define a set F_nat^* of arcs of net-token autonomous transitions in N^* as follows:

$$F^*_{nat} = \bigcup_{a_i \in F_i} \{(x'_i, y'_i \mid (x_i, y_i) \in F_i, x'_i \in P_{N^*}(x_i) \cup T^*_{nat}(x_i), y'_i \in P_{N^*}(y_i) \cup T^*_{sat}(y_i)\}. \quad (4.20)$$

The set F^*_{nat} gives all possible pairs of a place and a transition representing each net-token autonomous transition to be drawn in N^*. This is depicted in Figure 4.6 above for the example ERS. **Rule 5**: Generate family of transitions representing synchronisation

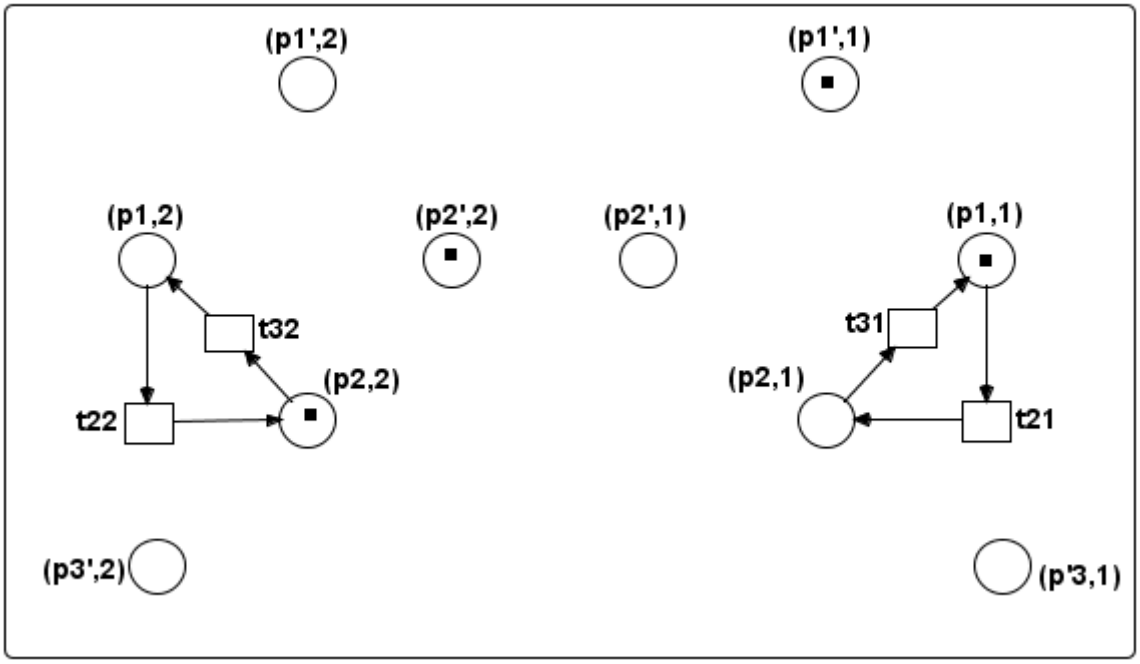

Figure 4.6.: Transitions and arcs generated after Rule 4

transitions obtained from the system net and in each net-token. An occurrence of a synchronisation firing mode presumes simultaneous occurrence of a transition $t \in \widehat{T}$ with a binding β in system net such that $\widehat{t} \in \ell$, and some net-tokens transitions $(t_1, \ldots, t_k) \in \ell$. Therefore, corresponding synchronisation transitions set of a P/T-net N^*, is composed of the number of synchronous transitions in each possible net-token referenced in the initial marking of the system net that can occur synchronously with the system net. This can be viewed as a combination of Rule 3 and Rule 4 with the condition that all involved transitions must be elements in the transition relation ℓ of an ERS.

Transitions $(t_1, \ldots, t_k)$ occur simultaneously with $\widehat{t} \in \widehat{T}$ of a system net, if $(\widehat{t}, (t_i, \ldots, t_k)) \in \ell$. We generate synchronisation transitions from an ERS in a P/T-net N^* accordingly. This implies that we will have $|\ell|$ such transitions in N^*. Each of these transitions is composed of a system net transition $t \in \widehat{T}$, and some transitions of net-tokens that participate

in synchronous firing of $\widehat{t}$. The family of these transitions sets is defined as follows.

$$T^*_{sync_i} = \bigcup_{i=1}^{k} \{t_{i.\beta_i(x)} = \{\widehat{t}, t_1, \ldots, t_k\} \mid x \in \omega(\widehat{t}), \widehat{t} \in \widehat{T}, t_1 \in T_1, \ldots, t_k \in T_k\}. \tag{4.21}$$

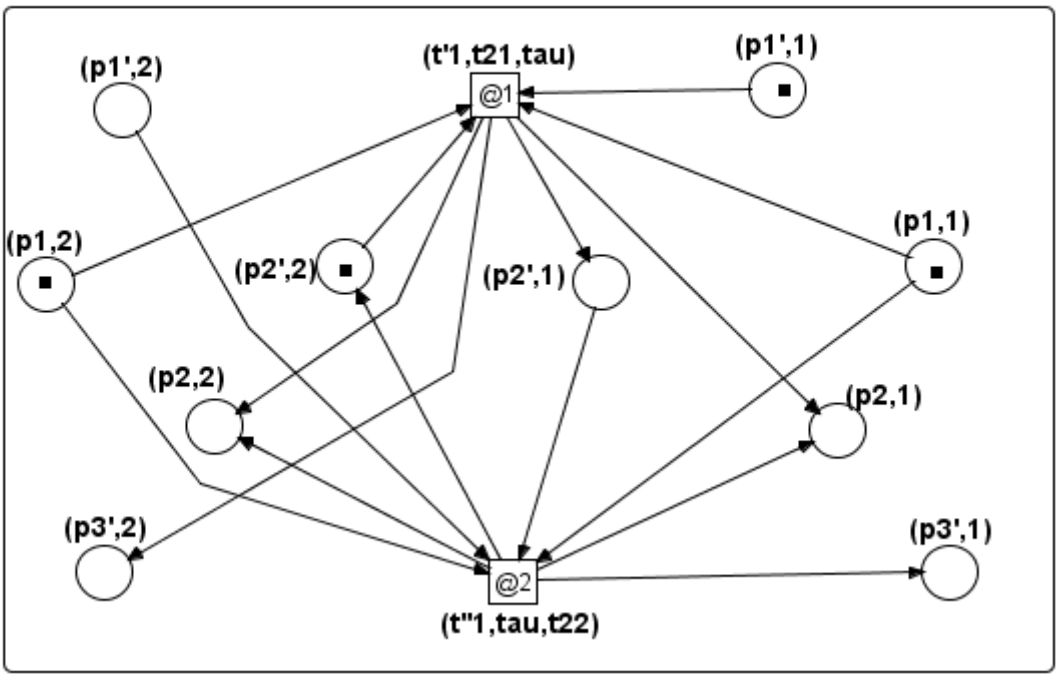

Figure 4.7.: Synchronous firing transitions and arcs

In our example two places $\widehat{p}_1$ and $\widehat{p}_2$ are marked with one net-token each in the initial marking. Therefore the number of object nets transitions that can occur simultaneously with $\widehat{t} \in \widehat{T}$ for every possible marking in the system net is $k = 2$. Thus we add two transitions $t_1 = \{\widehat{t}_1, t_{21}, \tau\}$ and $t_2 = \{\widehat{t}_1, \tau, t_{22}\}$ annotated with @1 and @2, which is shown in Figure 4.7 above.

The result transforming ERS into 1-safe P/T-net is shown in Figure 4.8 on page 67

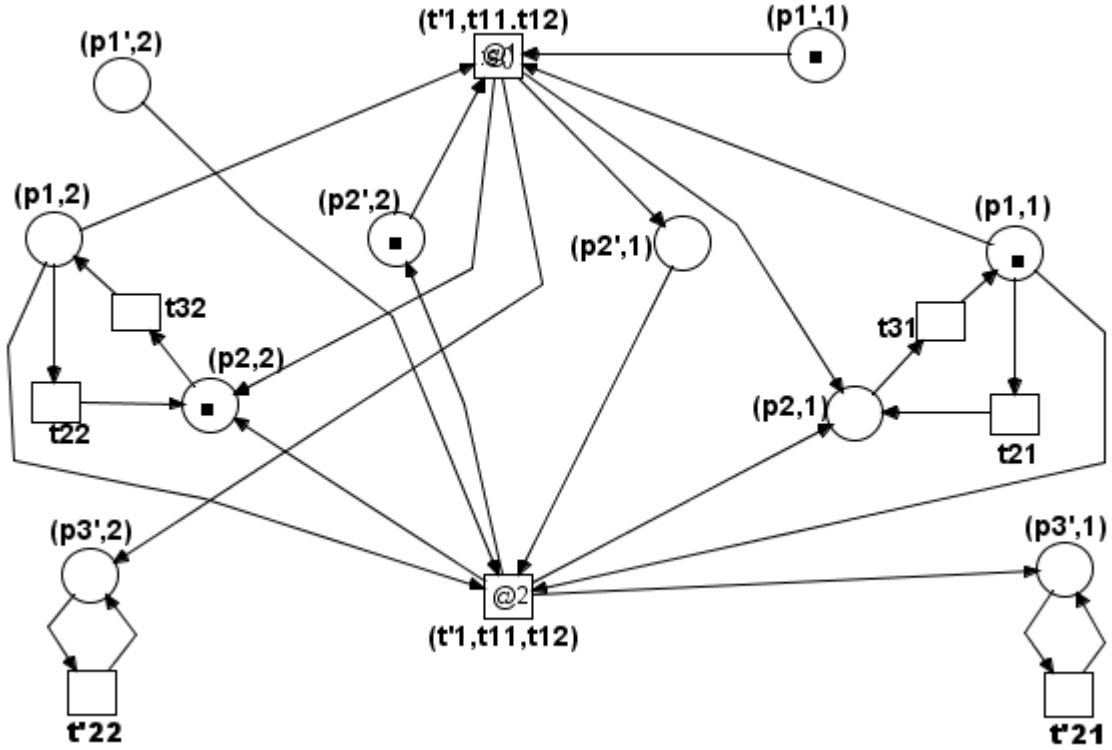

Figure 4.8.: Result of transforming ERS in 4.2 into 1-safe P/T net

4.4. Isomorphic Property of the State Spaces and Computational Complexity

4.4.1. Isomorphic Property of State Spaces

In this subsection we establish an isomorphism between the states of an *ERS* and the generated 1-safe P/T-net. Recall that in Rule 2 we defined two separate initial markings for the P/T-net N^* : $M_0^*(\widehat{p}, i)$ and $M_0^*(p, i)$. The former is an encoding of markings on places from the set of places $\widehat{P}$ of the system net in an *ERS* and the latter is an encoding of markings on places from the set of places P_i of a net-token i of an *ERS*. Likewise we defined three sets of transitions: T_{sat}^*, T_{nat}^*, and $T_{sync_i}^*$ from Rule 3, Rule 4 and Rule 5 respectively in N^*. In the following, we define some mappings from the P/T-net to an *ERS*.

Definition 4.9. *A mapping $\widehat{f}$ maps a marking M^* of a P/T-net N^* from the set of*

places $\widehat{P}$ to markings R of a system net of an ERS as follows:

$$\widehat{f}(M^*(\widehat{p},i)) = R(\widehat{p}) \, such \; that \, (\widehat{p},i) \in P^*_{N^*} : \widehat{p} \in \widehat{P} : i \in \mathbb{R}. \tag{4.22}$$

Definition 4.10. *A mapping f maps a marking M^* of a P/T-net N^* from the set of places P_i of net-token i of an ERS to a marking M of a net-token of an ERS as follows:*

$$f(M^*(p,i)) = M(p) \, such \; that \, (p,i) \in P^*_{N^*} : p \in P_i : i \in \mathbb{R}. \tag{4.23}$$

Definition 4.11. *$\widehat{g}$ is a mapping that maps a transition $t'_{\beta_i(x)} \in T^*_{sat}$ of P/T-net N^* to a system-autonomous firing mode $(\widehat{t},\tau) \notin dom(\ell)$ of an ERS as follows:*

$$\widehat{g}(t'_{\beta_i(x)}) = (\widehat{t},\tau), \tag{4.24}$$

where $\beta_i(x)$ is a binding function that binds a variable $x \in \omega(\widehat{t})$ on arcs adjacent to $\widehat{t}$ to an object net name.

Definition 4.12. *g is a function that maps a transition $t \in T^*_{nat}$ of P/T-net N^* to an object-autonomous firing mode $(\tau,t_i) \notin dom(\ell)$ of an ERS as follows.*

$$g(t) = (\tau,t_i). \tag{4.25}$$

Definition 4.13. *g_s is a mapping function that maps a transition $t_{i.\beta_i(x)} \in T^*_{sync_i}$ of P/T-net N^* to a synchronisation firing mode $(\widehat{t},t_1,\ldots,t_k) \in \ell$ of an ERS as follows:*

$$g_s((t_{i.\beta_i(x)}) = \{(\widehat{t},t_1,\ldots,t_k)\}. \tag{4.26}$$

With respect to these definitions, the following lemmas related to ERS and a P/T-net, which is constructed by applying Rules 1 to 5, hold.

Lemma 4.1. *For the initial marking at the system net level, the following equality holds:*

$$R^0(\widehat{p}) = \widehat{f}(M^*_0(\widehat{p},i)). \tag{4.27}$$

Proof. An initial marking of a system net in an ERS can be expressed by $R^0 = R^0(\widehat{p}), \forall \widehat{p} \in \widehat{P}$. By Rule 2, $(\widehat{p},i) \in P^*_{N^*}$ in the P/T-net has one token in the corresponding initial marking $M^*_0(\widehat{p},i)$, therefore $M^*_0(\widehat{p},i) = R^0(\widehat{p})$.

From Definition 4.9, $\widehat{f}(M^*_0)(\widehat{p},i)$ becomes $\widehat{f}(M^*_0)(\widehat{p},i) = R^0(\widehat{p}) = R^0(\widehat{p})$ □

Lemma 4.2. *Suppose that* $R = \widehat{f}(M^*)$ *and* $(\widehat{t}, \tau) = \widehat{g}(t'_{\beta_i(x)})$. *The following proposition holds:*

$$M^*[t'_{\beta_i(x)}\rangle \Leftrightarrow R[(\widehat{t}, \tau)\rangle \tag{4.28}$$

Proof. ($\Rightarrow$) Suppose that $t'_{\beta_i(x)} \in T^*_{sat}$ is a transition that represents an autonomous transition in the P/T- net then $(\widehat{t}, \tau) \in \widehat{T}$ is a corresponding transition in the system net. From $M^*[t'_{\beta_i(x)}\rangle$ and Definition 2.18, each place has at least $W^*_{sat}((\widehat{p}, i), t'_{\beta_i(x)})$ tokens namely for each place $(\widehat{p}, i) \in P^*_{N^*}$, the following inequality holds:

$$M^*((\widehat{p}, i)) \geq W^*_{sat}((\widehat{p}, i), t'_{\beta_i(x)}) \tag{4.29}$$

Since $R = \widehat{f}(M^*)$, the number of token in place $(\widehat{p}, i)$ equals the number of tokens in place $\widehat{p} \in \widehat{P}$ of a system net $\widehat{N}$:

$$M^*((\widehat{p}, i)) = R(\widehat{p}). \tag{4.30}$$

From Rule 3, the weight of the arc from $(\widehat{p}, i)$ to $t'_{\beta_i(x)}$ equals number of variables on the arc from $\widehat{p}$ to $\widehat{t}$ under the binding β:

$$W^*_{sat}((\widehat{p}, i), t'_{\beta_i(x)})) = \beta(\omega(\widehat{p}, \widehat{t})). \tag{4.31}$$

From 4.29, 4.30 and 4.31, for each place $\widehat{p} \in \widehat{P}$ the following holds:

$$R(\widehat{p}) \geq \beta(\omega(\widehat{p}, \widehat{t})). \tag{4.32}$$

From Definition 4.5, $R[(\widehat{t}, \tau)\rangle$,

($\Leftarrow$) (4.32) holds since $R[(\widehat{t}, \tau)\rangle$; (4.30) and (4.31) also hold. Therefore, (4.29) holds from Definition 2.18, and $M^*[t'_{\beta_i(x)})\rangle$ □

Lemma 4.3. *Suppose that* $R_1 = \widehat{f}(M^*_1)$, $M^*_1[t'_{\beta_i(x)}\rangle M^*_2$, *and* $R_1[\widehat{g}(t'_{\beta_i(x)})\rangle R_2$. *The following equality holds.*

$$R_2 = \widehat{f}(M^*_2). \tag{4.33}$$

Proof. Proof: From Definition 2.18, the number of tokens in place $(\widehat{p}, i)$ in a successor marking M^*_2 is expressed as follows:

$$M^*_2(\widehat{p}, i) = M^*_1(\widehat{p}, i) - W^*_{sat}((\widehat{p}, i), t'_{\beta_i(x)}) + W^*_{sat}(t'_{\beta_i(x)}, (\widehat{p}, i)). \tag{4.34}$$

Since $R_1 = \widehat{f}(M_1^*)$, (4.30) holds. Similarly to (4.31), it holds that

$$W_{sat}^*(t'_{\beta_i(x)}, (\widehat{p}, i)) = \beta(\omega(\widehat{t}, \widehat{p})). \tag{4.35}$$

Therefore:

$$M_2^*(\widehat{p}, i) = R_1(\widehat{p}) - \beta(\omega(\widehat{p}, \widehat{t})) + \beta(\omega(\widehat{t}, \widehat{p})). \, (\textit{See Definition. } 4.5) \tag{4.36}$$

Finally it holds that $R_2 = \widehat{f}(M_2^*)$ because (4.36) holds for each place. □

Lemma 4.4. *For the initial marking of the object net, the following holds:*

$$M^0(p) = f(M_0^*)(p, i). \tag{4.37}$$

Proof. An initial marking of an object net in an ERS can be expressed by $M^0 = M^0(p), \forall p \in P_i, i \in \mathbb{R}$ hold. Rule 2 says that place $(p, i) \in P_{N^*}^*$ in the P/T-net has one token in the corresponding initial marking $M_0^*(p, i)$, therefore $M_0^*(p, i) = M^0(p)$.

From Definition 4.10, $f(M_0^*(p, i))$ becomes $f(M_0^*(p, i)) = M^0(p)$. □

Lemma 4.5. *Suppose that $M = f(M^*)$ and $(\tau, t_i) = g(t)$. The following proposition holds:*

$$M^*[g(t)\rangle \Leftrightarrow M[(\tau, t_i)\rangle. \tag{4.38}$$

Proof. ($\Rightarrow$) Suppose that $t \in T_{nat}^*$ is a transition that represents an autonomous transition in the P/T- net then $(\tau, t_i) \in T_i$ is a corresponding transition in the object net. From $M^*[t\rangle$ and the Definition 2.18, each place has at least $W_{nat}^*((p, i), t)$ tokens namely for each place $(p, i) \in P_{N^*}^*$, the following inequality holds:

$$M^*((p, i)) \geq W_{nat}^*((p, i), t). \tag{4.39}$$

Since $M = f(M^*)$, the number of tokens in (p, i) equals the number of tokens in $p \in P_i$ of an object net N_i

$$M^*((p, i)) = M(p). \tag{4.40}$$

From Rule 4, the weight of the arc from (p, i) to t equals the weight of the arc from p_i to t_i

$$W_{nat}^*((p, i), t) = W_i(p_i, t_i). \tag{4.41}$$

From (4.40) and (4.41), for each place $p \in P_i$ the following inequality holds:

$$M(p) \geq W_i(p_i, t_i). \tag{4.42}$$

From Definition 4.6, $M[(, t_i)\rangle$.

($\Leftarrow$) (4.42) holds since $M[(\tau, t_i)\rangle$; (4.40) and (4.41) also hold. Therefore, (4.39) holds. From Definition 2.18, $M^*[t\rangle$. □

Lemma 4.6. *Suppose that $M_1 = f(M_1^*)$, $M_1^*[t\rangle M_2^*$, and $M_1[g(t)\rangle M_2$. The following equality holds:*

$$M_2 = f(M_2^*) \tag{4.43}$$

Proof. From Definition 2.18, the number of tokens in place (p, i) in a successor marking M_2^* is expressed as follows:

$$M_2^*(p, i) = M_1^*(p, i) - W_{nat}^*((p, i), t) + W_{nat}^*(t, (p, i)). \tag{4.44}$$

Since $M_1 = f(M_1^*)$, (4.40) holds. Similarly to (4.41), it holds that

$$W_{nat}^*(t, (p, i)) = W_i(p_i, t_i). \tag{4.45}$$

Therefore, the following equation holds:

$$M_2^*(p, i) = M_1(p_i) - W_i(p_i, t_i) + W_i(t_i, p_i) = M_2^*(p, i)\,(\textit{See Definition. 4.6}) \tag{4.46}$$

Finally it holds that $M_2 = f(M_2^*)$ because (4.46) holds for each place. □

Lemma 4.7. *Suppose that $(R_1, M_1) = f_s(M_1^*)$ and $t_s = g_s(t_i.\beta_i(x))$. The following proposition holds:*

$$M_1^*[g_s(t_i.\beta_i(x))\rangle \Leftrightarrow (R_1, M_1)[t_s\rangle. \tag{4.47}$$

Proof. ($\Rightarrow$) For $\widehat{t}$, it can be proven in a similar way to Lemma 4.2 that

$$\forall \widehat{p} \in {}^\bullet\widehat{t} : R(\widehat{p}) \geq \beta(\omega(\widehat{t}, \widehat{p})). \tag{4.48}$$

For $(t_1, \ldots, t_k)$ it can be proven in a similar to Lemma 4.4 for each net-token transition $t_i \in T_i$ that

$$\forall p_i \in {}^\bullet t_i : M_1(p_i) \geq W_i(p_i, t_i). \tag{4.49}$$

From Rule 5, and equations (4.48) and (4.49) it holds that $(R_1, M_1)[t_s\rangle$. ($\Leftarrow$) For $t_i.\beta_i(x) \in T_{sync_i}^*$ which is added in Rule 5, it can be proven that in a similar way to

Lemma 4.2 that

$$\forall(\widehat{p}, i) \in P'_{N^*} : M_1^*(\widehat{p}, i) \geq W^*((\widehat{p}, i), \widehat{t}). \tag{4.50}$$

Similarly, it can be shown from Lemma 4.4 for $t_i \in T_i$ that participate in $t_i.\beta_i(x) \in T^*_{sync_i}$ that

$$\forall(p_i, i) \in P_{N^*} : M_1^*(p_i, i) \geq W^*_{nat}(p_i, t_i). \tag{4.51}$$

$(\widehat{t}, t_1, \ldots, t_k)$ share no input places by assumption in Rule 1. From Definition 2.18, (4.50) and (4.51): $M_1^*[t_i.\beta_i(x)\rangle$. □

Lemma 4.8. *Suppose* $(R_1, M_1) = f_s(M_1^*), M_1^*[t_i.\beta_i(x)\rangle M_2^*$ *and* $(R_1, M_1)[g_s(t_i.\beta_i(x)\rangle\,(R_2, M_2)$. *The following equality holds:*

$$(R_2, M_2) = f_s(M_2^*). \tag{4.52}$$

Proof. It can be proved in a similar way to Lemma 4.3 and 4.6 by Definition 2.18, and Rules 3 and 4. □

From the above Lemmas, the following theorem holds.

Theorem 4.2. *Let RS be a 1-safe ERS. Let also N^* be a 1-safe P/T-net obtained from RS by the set of transformation Rules 1 to 5 above. Then state spaces of RS and N^* are isomorphic.*

Proof. Lemmas 4.1 and 4.4 defines a one-to-one mapping between the initial markings of the 1-safe P/T-net N^* and the initial marking in RS. From Lemma 4.2 a system-autonomous firing mode $(\widehat{t}, \tau)$ is enabled in a marking (R, M) if, and only if, the corresponding transition $t'_{\beta_i(x)}$ is enabled in the corresponding marking M^*. Also from Lemma 4.5 an object-autonomous firing mode (τ, t_i) is enabled in a marking (R, M) if, and only if, the corresponding transition t is enabled in the corresponding marking M^*. Again, from Lemma 4.7 a synchronous firing mode $(\widehat{t}, t_1, \ldots, t_k)$ is enabled in a marking (R, M) if, and only if, the corresponding transition $t_i.\beta_i(x)$ is enabled in the corresponding M^*. Finally from Lemmas 4.3, 4.6 and 4.8, the generated markings in the 1-safe P/T-net can be mapped to the generated markings in the RS. □

Thus we have shown that every Elementary Reference-net System (ERS) can be transformed to behaviourally equivalent 1-safe P/T-net. Hence the standard analysis techniques for 1-safe P/T-net can be applied for a transformed ERS. In the next subsection we discuss the computational complexity associated with transforming an ERS into 1-safe P/T-net.

4.4.2. Computational Complexity Result for Transformation

In this subsection we discuss the computational complexity associated with translating *ERS* into a 1-safe P/T-net. Let us suppose that the number of adjacent places for transition at the system net level is at most ρ, the number of places for each transition of object nets is at most γ, the number of object nets is k, and the number of to tokens in the initial marking of the system and object nets is δ

The complexity of generating places at Rule 1 is $\mathcal{O}(k|\widehat{P}| + |P_\Sigma|)$ because at most k copies are generated for each place of the system net, and at most $|P_\Sigma|$ are generated for each place of object nets. The complexity of generating initial marking at Rule 2 is $\mathcal{O}(\delta)$. The number of binding for each arc variable adjacent to each transition of the system net is at most k^ρ; therefore, the complexity of generating transitions at Rule 3 is $\mathcal{O}(k^\rho|\widehat{T}|)$. The number of arcs for each transition is at most ρ; therefore, the complexity of generating arcs at Rule 3 is $\mathcal{O}(k^\rho\rho|\widehat{T}|)$. The complexity of generating transitions at Rule 4 is $\mathcal{O}(|T_\Sigma|)$. The number of arcs for each transition of object nets is at most γ; therefore the complexity of generating arcs at Rule 4 is $\mathcal{O}(\gamma|T_\Sigma|)$. One transition representing a synchronous transition is composed of $k+1$ transitions; therefore, the number of transitions representing synchronous transition sets grows as the k power of the number of transition in the *ERS* this is because the number of object nets that can synchronise with the system net is at most k. Consequently, the complexity of generating transitions at Rule 5 is $\mathcal{O}(k^{k\rho}|\widehat{T}|)$.

In the worst case, ρ and γ equals the number of places and k grows as the number of object nets increases. Thus, the computational complexity of complete transformation is exponential with the size of the *ERS*. Moreover, because of conditions (1), (2), (3) and (4) imposed on variables appearing on arcs adjacent to a transition $\widehat{t}$ of the system net in Definition 4.2, k does not change. Therefore, the time required for the analysis of the resulting 1-safe P/T-net will be huge as far as the overall efficiency is concerned.

4.5. Chapter summary

While the general elementary object systems (EOS) are suitable formalism for modelling agent behaviour, communication and mobility they come with some constraints in its definition that limit their expressiveness for automatic verification purposes. Firstly, arc inscriptions are not associated to an arcs of a safe-EOS model; making it impossible to

know exactly which net tokens to remove from the input places of a system net transition and which ones to add to its output places when it fires. Secondly, the firing rule uses a so called distributed token semantics in which the tokens of an object net may be distributed if copies of that object net are created during firing. Consequently, this firing rule incorporates the possibility to test if an object net token is an empty marking this resulted in making EOS to be considered a Turing-complete formalism. In this thesis, a modification to the definition of the general elementary object net which relaxes these constraints is defined under the name elementary reference-net systems. To overcome the arc inscription constraint, we tackled it in two ways: Firstly, we provide each marked object net which are located in places of the system net with unique names so that object nets with the same marking can be distinguished. Secondly, we defined variables labelling arcs of the system net taken from a finite set variables that are bound to object nets names when firing system net transitions instead of statically typing of the system net places. Also, we define some structural restrictions to ensure that new object nets can neither be created nor an existing one destroyed after a transition firing in the system net. Again, we define dynamic restriction by extending the notion of 1-safe P/T nets to ERS to guarantee that the state space is finite and show that the state space of ERS is bounded. Most importantly, we proposed a set of rules for transforming ERS to behaviourally equivalent 1-safe P/T net in such a way that established tools can handle important questions regarding the verification of various properties. Furthermore, we established clearly, an important relation between the isomorphic properties of state spaces of 1-safe ERS and 1-safe P/T net. Among such results are the established Lemmas, and prove of a theorem which related the state of 1-safe P/T nets and state space of a 1-safe ERS. Moreover, we adopt a different way of introducing the procedure for transforming a class nets-within-net formalism into 1-safe P/T net, which result, in a neater and easier-to-understand presentation compared to some related studies. In particular we used not only algorithm-dependent proofs but also, mathematical proofs. Finally, we established the computational complexity for transforming safe-ERS into 1-safe P/T net which confirms the effectiveness of the method.

The need for a definition of an *elementary reference-nets system, ERS*, arose when it was important to use partial order (unfolding) approach for dynamic analysis of EOS which is a class of nets-within-nets paradigm. Thus, in the next chapter, we present how all developed algorithms were implemented and collected into a software tool.

Chapter 5.

Implementation of Transformation Rules

This chapter describes the ERStoPTnet, a software tool for transforming Elementary Reference-net System into a P/T net. ERStoPTnet is developed in Java and should be considered a prototype transformation tool for ERS into P/T net using the set of transformation rules presented in Chapter 4. This tool may be applied to large examples of ERS.

Section 5.1 presents an overview of the implementation. How ERS can be specified in RENEW and the description of the input language of ERStoPTnet is given in Section 5.2, while Section 5.3 provides some implementation details. Example of ERStoPTnet usage is presented in Section 5.4.

5.1. Overview

ERStoPTnet is a transformation software tool for transforming a class of nets-within-nets to a P/T, it takes PNML format of this class of net, transforms it into a low-level net and produces two kinds of output formats: a net in PNML format that can be exported to RENEW and a net in a textual representation for use with a Petri net unfolder. The Transformation of ERS are described in ERStoPTnet using the PNML generated in RENEW (see Section 5.2).

The structure of implementation of the entire development of ERStoPTnet is depicted

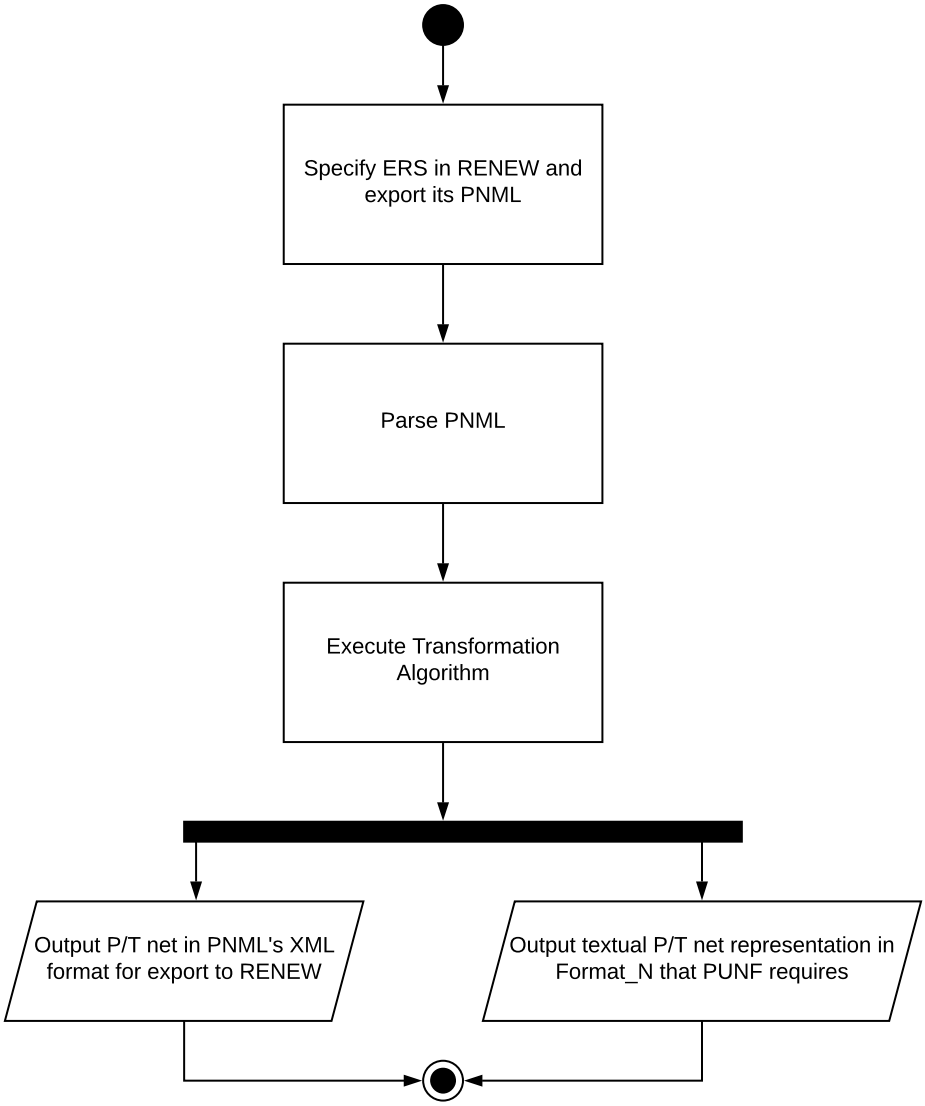

Figure 5.1.: Work flow overview

in Figure 5.1. The second and the third steps, in the figure, are performed automatically upon invocation of the tool.

The ERS formalism serves as the starting point of our implementation. It was described in Chapter 4.

- In step 1 the ERS model is drawn using the graphical editor of RENEW tool, and subsequently, exported in a PNML format. The PNML file includes the XML description of the system net and the net-tokens to be parsed as input to the ERStoPTnet. At this stage RENEW automaically builds the PNML representation of all nets involved in the model and their elements (such as net names, places, arc, transitions, arcs inscriptions, etc.)

- In step 2 all the class diagram with all the features of the Elementary Reference-net System is modelled as Java Ecore model in Eclipse. Figure 5.2 shows the class diagram with all the features of an ERS. This diagram refers to all objects which are defined by RENEW in the PNML model: these are the net, place, arc and transition classes respectively.

- In step 3 the PNML file is parsed using standard Java application programming interface SAX (Simple API for XML), which is an event-driven algorithm for parsing XML documents. At this stage ERStoPTnet builds an internal representation of ERS elements appearing in the input file using data structure such as Lists of Lists, Sets, and Maps.

- In step 4 ERStoPTnet automatically executes the transformation algorithms and generate two different output formats: one of the outputs is a P/T net in PNML format for export to RENEW, which can be used for simulation purpose and to show the practicality of Theorem 4.2, and the other output is a textual P/T net representation in Format_N (Format_N Khomenko (2012) the agreed upon file format for low-level (LL) used as input for the PUNF-toolset.

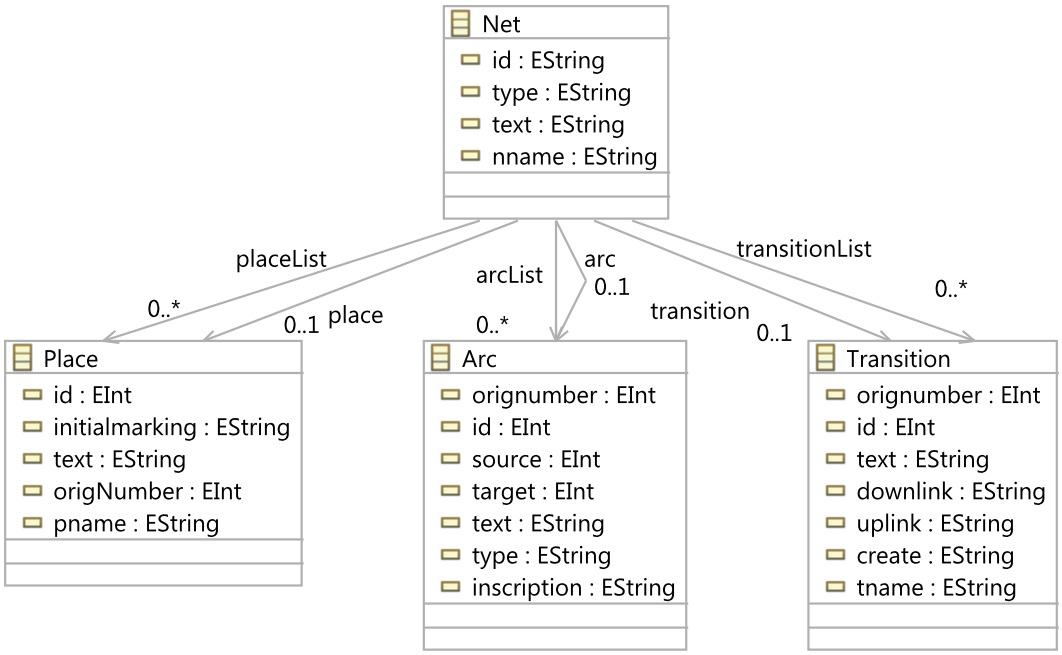

Figure 5.2.: Ecore model for ERS net type as diagram

5.2. The input language PNML

As mentioned in Section 2.4, PNML (Petri Net Markup Language) is an XML transfer format for Petri nets. So, PNML defines how these graphical Petri net concepts are saved or represented in XML. This is achieved by mapping every concept or feature of the net model to some XML construct. Appendix A shows a simple library scenario example of an ERS that models a system net and two net-tokens in its graphical representation (concrete syntax) specified in RENEW (some labels have been omitted for clarity); Appendix B, Listing B.1 shows an excerpt of its corresponding representation in PNMLs XML-format.

Note that the listing also shows an example of how the PNML functionality of RENEW can allow multiple nets to be contained within one PNML file: line numbers 2 - 62 contains net elements from the system net, and line numbers 63 - 97 contains elements from nettoken1 while lines 98 - 132 contain those elements from nettoken2

PNML is the input language of ERStoPTnet. A PNML file of an ERS spefied in RENEW, describes all the components of ERS following closely the formalism presented in Section 4.2.1 and 4.2.2.

5.2.1. General structure of the PNML file

The PNML file is composed of mainly four components/sections which include: *net*, *places*, *transitions* and arcs, and these objects can have some kind of *label*. Also, it defines all kinds of graphical information that can be attached to the different objects, such as position, dimension, fill color, and line color.

1. Net declarations. In this section of PNML nets are defined using a sequence of properties, each of which has the following syntax:

```
1 <net id="any alphanumeric value" type="RefNet">
2    <net_body>
3 </net>
```

   where < net id ="any alphanumeric value" type="RefNet" > is a valid net identifier and type (see for example, line 2 in Listing B.1), and < *net_body* > contains the declaration of places, transitions, and arcs for each net.

2. Place declarations. In this section the declaration of a place is defined as follows:

```
1 <place id="any non-negative integer">
2      <place_body>
3 </place>
```

   where < place id="any non-negative intege" > is a valid place identifier, and < *place_body* > contains the declaration of the initial marking, the place name, and their graphical information for each place. If a place has no initial marking, the < *place_body* > will contain only the name of the place and it's graphical information.

3. Transition declarations. In this section the set of transitions is defined by the syntax:

```
1 <transition id=" nay non-negative integer">
2      <transition_body>
3 </transition>
```

   where < transition id="any non-negative intege" > is a valid transition identifier, and < *transition_body* > contains the declaration of the transition name, transition inscriptions, and their graphical information for each transition. Communication between the system net and net-tokens within the Elementary Reference-net formalism is handled via synchronous channels, based on the concepts described in

Chapter 4. Synchronous channels connect two transitions during firing. Transitions inscribed with a synchronous channel can only fire synchronously with another transition inscribed by matching channel, meaning that both transitions involved have to be activated before firing can happen. The inscription of the system net transition that initiates the firing is called the *downlink*. The downlink must know the name of the net-token in which the other transition, the so-called *uplink*, is located. Therefore, the inscription of the downlink has the form *net-tokenname:channelname()*. The uplink's inscription is similar, but does not contain a net name, so that it has the form *:channelname()*. Uplinks are not exclusive to one downlink and can be called from multiple downlinks, so that it is possible to inscribe one system net transition with multiple downlinks, to enable it fire simultaneously with synchronous transitions from different net-tokens.

Figure A.1 shows a simple example of an ERS. The example was modelled using the RENEW tool. It consists of three kinds of nets: the system net, and two net-tokens. The two net-tokens both possessed the same basic structure, but use different names. The system net functions as a kind of container for the net-tokens. The system net transition inscribed **x:new netToken1; y:new netToken2** initiate the creation of new netToken1 and netToken2 when it fires. These new created nets are put onto the output places p_2 and p_4 of the transitions respectively. Based on the premise that ERS formalism uses reference semantics, it means that tokens within net places do not (exclusively) correspond to instances of these net-tokens, but only reference net-token names. Hence a $< transition\_body >$ declaration in the PNML, may consist of inscriptions for *create*, *downlink* and *uplinks*. See for example, lines 19 - 26 in Listing B.1 shows the body of a transition with the **create** inscription, and lines 48 - 58 shows the body of a transition with *downlink* inscription while lines 86 - 93 shows the body of a transition with *uplink* inscription).

4. Arc declarations. In section the declaration of an arc is defined as follows:

```
1 <arc id="any non-negative integer" source ="source node id" target="
    target node id">
2     <arc_body>
3 </arc>
```

where < arc id="integer"source ="node id" target="node id" > contains a valid arc identifier, a source node (place/transition) identifier and a target node (place/-transition) identifier respectively depending on whether the source or target node is a place or a transition. As discussed in Section 4.2.1 in the system net, arc adjacent to a place may carry inscription of a variable that will be bound to a net name in

that place. Thus < *arc_body* > contains the declaration of the arc inscription and arc type and their graphical information for each arc.

5.3. Implementation details

The source code of ERStoPTnet has been structured in separated packages, with a certain number of shared variables. These shared variables include the internal representation of the PNML input and encodings for nets, places, transitions, arc, etc. The source code of ERStoPTnet includes various classes each of which corresponds to a specific package:

- **com.xml.parser.file**: this package includes the classes defining the components of the PNML document of the specified ERS i.e., each net involved in the model together with their places, transitions and arcs. These classes includes Java code to build the internal representation of the PNML as data structures.
  - the class **Net** contains:
    - net attributes
    - place list
    - arc list
    - transition list
    - constructors
    - with corresponding getters and setters methods and the toString() method to return string representation of data elements.
  - the class **Place** contains:
    - place attributes
    - constructor
    - with corresponding getters and setters methods and the toString() method

  - the class **Arc** contains:
    - arc attributes
    - constructor
    - with corresponding getters and setters methods and the toString() method
  - the class **Transition** contains:
    - transition attributes
    - constructor
    - with corresponding getters and setters methods and the toString() method

- **com.xml.parser.refnet**: In this package we have three additional Java classes, which are written completely manually. One implements the Java Architecture for XML Binding (JAXB), for converting all the transformed net objects into a XML file which was discussed in step:4 of the implementation over. The other two are convenience classes, which make it easier to implement our parser and transformer. We briefly discuss them below:

  - The class **MyHandler**: this class includes the SAX API defining the formal parser for PNML. Although the SAX parser is a Java API for sequential reading of XML files, but XML format of PNML requires recursive reading of the PNML file due to the possibility that some system net synchronous transitions might have multiple downlinks channels. Therefore, we include java code to build the internal representation of PNML file containing such multiple channels as a list.

  - The class **Transformer** is the main class. This class contains the method calls to parse input PNML file, to perform setup operations at the beginning of a run, perform transformation operations, to perform output operations and close all files upon completion of transformation.

5.4. Usage

ERStoPTnet is implemented entirely in Java to secure the platform independence and provides an elegant, easy-to-use graphical user interface that allows for the loading, transformation of Elementary Reference-net System into low level Petri nets, saving the result into PNML file and textual representation of low-level nets for interchange among Petri net tool. The source code is available from Abdullahi and Müller (2016).

ERStoPTnet is created as a runnable JAR file. No special installation is required. Just copy the JAR file into a folder. It is run by simply double clicking on the executable. One important point to note is that the PNML file that it takes as input must be in the same folder.

ERStoPTnet needs just one input parameter: the name of the PNML file to be transformed and one output parameter for the name of output file to be created for storing the textual representation of the net in Format_N for the transformed net. Listing 5.1 shows an excerpt from the output in Format_N.

Listing 5.1: Excerpts from ERStoPTnet output in Format_N

```
1 PEP
2 PTNet
3 FORMAT_N
4 %———Places——
5 PL
6 1"p'7"
7 2"p'4"M1
8 3"p'3"
9 4"p'6"
10 5"p'2"M1
11 6"p'5"
12 7"p'7"
13 8"p'4"M1
14 9"p'3"
15 10"p'6"
16 11"p'2"M1
17 12"p'5"
18 13"p1"M1
19 14"p2"
20 15"p3"
21 16"p1"M1
22 17"p2"
23 18"p3"
```

```
%——Transitions——
TR
23" t '5"
24" t '6"
25" t '5"
26" t '6"
33" t1"
34" t1"
45" t '2"
46" t '2"
```

In step 4 of Section 5.1 we mentioned that one of the outputs of ERStoPTnet is a P/T net in PNML format for export to RENEW, which can be used for simulation of resulting net. Figure 5.3 on page 84 gives a corresponding P/T net of the ERS from Figure A.1 exported into PNML format to RENEW. However, since simulation as an analysis technique is not exhaustive, we focus on verification of resulting net which we present in the next chapter using the textual P/T net output represented in Format_N.

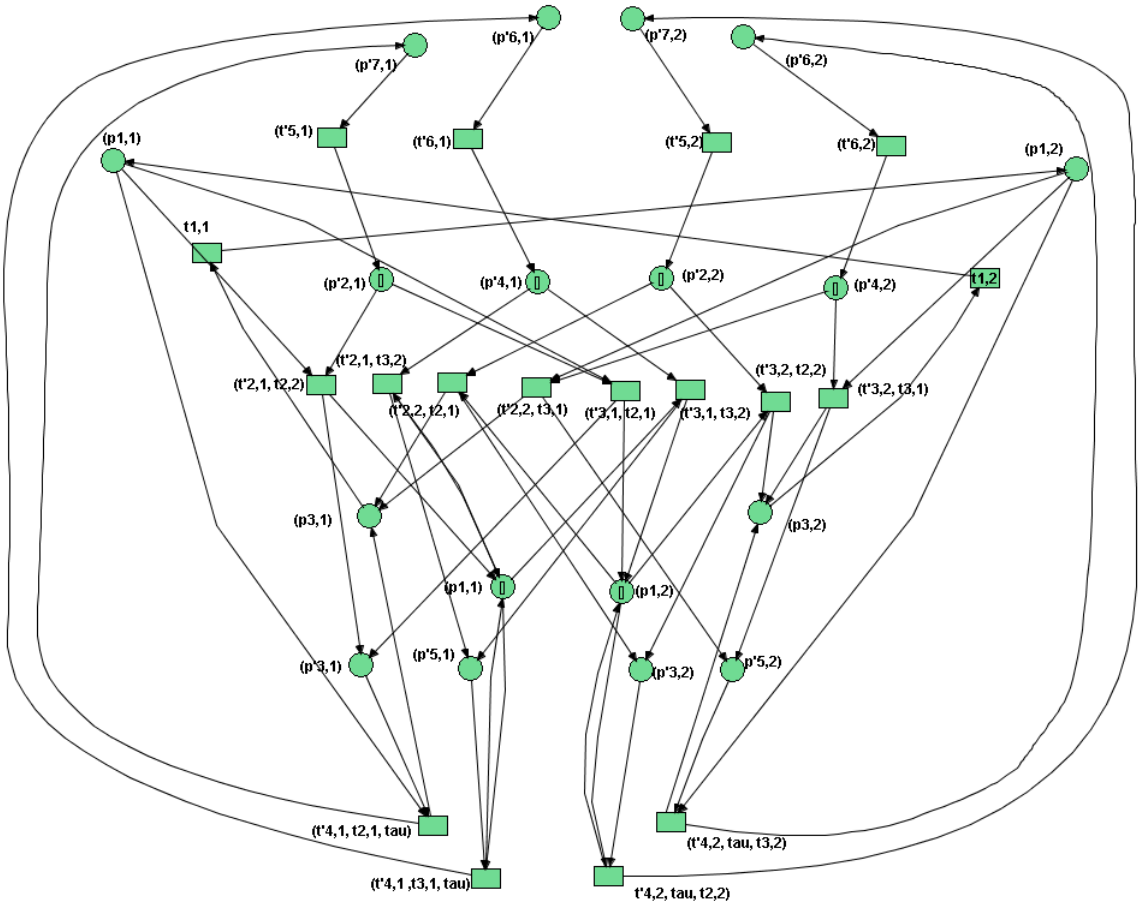

Figure 5.3.: P/T-net corresponding to output PNML format of the ERS shown in Appendix A.1 exported to RENEW

Chapter 6.

Verifying Transformed ERS and Bounded Petri net Models

Recall that the aim of this thesis was to apply formal verification techniques on the resulting low-level net obtained after transforming the Elementary Reference-net System (ERS). Theoretical results about the transformation of ERS (algorithm and complexity) were described in chapter 4 and the implementation of the developed tool that permits the automatic transformation was described in chapter 5. This chapter presents verification by using *prefixes of unfoldings* on some example P/T-nets which are output in format_N by the transformation tool.

More specifically, the main contributions described in this chapter are:

1. The translations into Boolean satisfiability formulae of the problems of deadlock and reachability checking using finite complete prefixes, are devised.

2. The development of a generic tool called PrefixtoCNF. This tool parses a file containing the description of a finite complete prefix of low-level Petri nets in particular of a transformed ERS, and any bounded Petri net model in general, and subsequently generates an encoding for deadlock problems into a CNF.

3. The implementation of the above mentioned translations in the tool is presented, with experimental results to support the feasibility of our approach.

However, similar to other concurrent formalisms which are modelled as Petri nets, it is apparent that application of static reachability analysis for automatic verification of the Petri net representing a transformed ERS, will also suffer from the so called state

space explosion problem (SSE). Moreover, concurrency is not only the source of the SSE problem in the transformed ERS. Another important source is the synchronisation between the involved nets contained in such model. Consequently, the size of the resulting transformed net can grow very quickly since it is bounded above by the product of the set of the system net transitions and sets of transitions of each individual net as shown in section 4.4.2. Interestingly, It has been proven in Heljanko (1999) that model checking a fixed size CTL formula on finite complete prefixes of unfolding is PSPACE-complete. Hence we sort to use *bounded model checking with SAT solver via prefixes of unfolding* to detect a simple and important concurrency property (deadlock) on resulting transformed nets.

Basically, bounded model checking with SAT via prefixes of unfolding of Petri net, requires three steps: In the first step, the prefixes are generated. In the second step, the sequential behaviour of a prefix of the unfolding over a *configuration* for some properties to be verified is encoded as a Boolean satisfiabilty formulae. In the third step, these formulae in *Conjunctive Normal Form (CNF)* popularly known as DIMACS format (DIMACS Challenge (1993), a format named after the Centre for Discrete Mathematics and Computer Science, currently at Rutgers University)) is given to a propositional decision procedure, i.e., a SAT solver, to either obtain a satisfying assignment or to prove there is none.

We have chosen to use an existing tool for unfolding the resulting Petri nets derived after automatically transforming ERS models, instead of developing this functionality from scratch. This tool is *PUNF* which we described in Section 6.1.2. It provides the desired prefixes of unfolding for bounded Petri net in an efficient and accessible way.

To be able to encode properties to be verified as Boolean satisfiability formula, we devised translations for the problem of deadlock and reachability checking into Boolean satisfiabilty formulae on prefixes of the unfolding and also, developed a generic tool which can automatically generate the DIMACS format file. Subsequently, we present how we technically model checked and evaluated the resulting low level Petri nets and provide experimental results. In addition, we discuss comparative analysis results we obtained by evaluating our results with other results from a well established technique that uses prefixes of unfolding in deadlock detection: the *constraint-based logic programming* technique by Heljanko (1999). Although our evaluation is narrow in scope, being restricted to deadlock checking we believe that this basic approach is broadly applicable since deadlock freedom is usually an important goal when developing and verifying a concurrent system. Moreover, many safety properties can be reduced to verifying deadlock freedom of modified systems Godefroid and Wolper (1991). Furthermore, any one conducting

comparative analysis with technique that uses prefixes of unfolding as a compact data structure for automated verification would benefit from our experience.

The rest of the chapter is structured as follows: we review automated verification of concurrent systems, the state space explosion problem, approaches for combating this problem and an existing tool used for unfolding in Section 6.1. Section 6.2 describes notions related to net unfoldings. In Section 6.3 we describe some Petri net benchmark models, which are applied to test the performance of the developed tool. For the comparative analysis in Section 6.6.2 we used the same Petri net benchmark models. In Section 6.4 we give an overview of Boolean satisfiability and its properties. Section 6.5 presents the translation of deadlock detection problem into SAT formulae. Section 6.6 presents the implementation results of the experiments and draws some conclusions about the relative strengths and weaknesses of our technique. Section 6.7 presents the chapter summary.

6.1. Automated Verification of Concurrent Systems

This section discusses verification of concurrent systems and presents a brief review of some approaches to tackle the state explosion problem. Although, among these approaches, we chose to present a brief description of the *unfolding* technique and existing tools for constructing the prefixes of unfolding safe Petri nets. This choice is motivated by a study in (James and Roggenbach, 2011) and (Esparza and Heljanko, 2008) which show that the *prefix* of the unfolding is more suitable for *bounded model checking* which is the approach we applied in our choice of verification technique.

When a concurrent system is modelled as a Petri net the reachability analysis provides a technique for automatically exploring and detecting most concurrency errors. However, verification techniques for systems that exhibit high level of concurrency have to deal with the additional complexity introduced by concurrency. For instance, if two concurrent acting processes P and Q are present, the verification technique needs to explore, in principle, the execution path where the order of the execution of P and Q, is such that P happens before Q, or Q happens before P, or even when P and Q happen simultaneously. It then means that if k concurrent acting processes are present, the number of execution paths will grow exponentially with k. Concurrency is widely known as the main source of the *state-space explosion* SSE problem for model checking.

6.1.1. Approaches for Tackling SSE and Tools

In automated verification of concurrent systems, approaches which offer powerful memory reduction while retaining a capacity for verification that mitigate the state-space explosion problem exit and are mainly classified into: symbolic model checking or partial-order model checking methods. Customarily, automated verification explicitly generates the entire reachability graph of the model. On the contrary, symbolic model checking explores and manipulates sets of reachable states represented implicitly, (e.g., in the form of ordinary binary decision diagram, or BDD). Symbolic model checking was introduced by Ken McMillan (McMillan, 1992) who used BDDs as a data structure to store and manipulate set of states implicitly.

The Partial-order approach comprises two distinct directions of similar nature often presented differently: the first one, partial-order reduction methods (Valmari 1988; and Peled 1993). They work by classifying all execution sequences of the system according to some equivalence relation such that all sequences in each equivalence class either satisfy or violate the property. Then they explore at least one trace on each equivalence class, thus, removing the sequences which are redundant for the trace representation.

The second approach aims to represent the partial-order semantics of Petri nets directly by transforming a Petri net transition firings into a set of events partially ordered by precedence relation. Such a transformation is called an unfolding since in every such set any transition (and thus any place) may be represented several times within a process according to possible firing of the transitions. In this representation, concurrent events are modelled by independent transitions, thus featuring any of their ordering sequence without representing them, a fact which the model checker uses to avoid individually generating each of them. There are several works that have enhanced this approach presenting efficient algorithms for constructing such compact structure and verifying system properties on it (e.g., McMillan (1992), and Esparza et al. (1996)).

Since the beginning of the 21st century, the algorithm for constructing prefixes of unfolding have been greatly improved (Esparza and Heljanko (2008); Esparza et al. (2002); Khomenko and Koutny (2001) Esparza and, Heljanko, 2008). Initially, algorithms for constructing complete prefixes were developed for ordinary Petri nets, currently, it has been extended to unbounded Petri net Abdulla et al. (2000), high-level Petri nets (Khomenko et al. (2003) (b)), Contextual nets Baldan et al. (2008); and Rodríguez (2013)). This algorithm has been implemented in several studies: (Esparza and Heljanko, 2001; Schröter, Schwoon, and Esparza, 2003; Schröter, and Khomenko, 2004; König and Kozioura (2008)

König and Kozioura 2005), and also applied to analysis and synthesis of asynchronous circuits in Khomenko et al. (2004), applied to monitoring and diagnose of discrete event systems in (Benveniste et al., 2003; Chatain and Claude 2004; Grabiec et. at, 2010) and analysis of asynchronous communication protocols in (Yu Lei and Purushothaman 2005).

6.1.2. Unfolding tools

There are existing tools for unfolding nets available including Mole , PUNF and Cunf.

1. Mole, (Schwoon and Römer, 2016) which is maintained by Stefan Schwoon, implements the Esparza\Römer\Vogler (SVR) unfolding algorithm for low-level Petri nets.

2. PUNF, (Khomenko, 2016) is maintain by Victor Khomenko, it is a suite of tools including MPSAT, PCOMP and MP2DOT. It can automatically construct complete finite prefixes of bounded Petri nets in efficient manner whenever it is given as input, a textual representation of the net in FORMAT_N.

3. Cunf, (Rodríguez and Schwoon, 2013) is an unfolding tool for Petri nets with read arcs, developed by César Rodríguez.

6.2. Branching process, Configurations and Cuts

A P/T-net system can be "unfolded" into a labelled *occurrence nets*, a subclass of nets with a particularly simple, tree-like structure. The nodes of the occurrence net are labelled with the places and transitions of the original net; Therefore, a reachable marking of an unfolding can always be understood in the context of the original net using the labels. The construction of an unfolding starts with each place that are contained in the set of initial marking of the original net. If in the current occurrence net some reachable marking enables a transition t then a new transition labelled with t and a new place labelled with output place of t are added to the occurrence net. Thereafter, edges are drawn to connect the newly added transition t to the set of its input places and to the set of its freshly added output places. The nets (a) and (b) presented in Figure 6.2 on page 92 are constructed in this way. The labelled occurrence nets obtained through unfolding of

a net are called branching processes. Although, the construction of an unfolding can be infinite, the unfolding process can be stopped at different times yielding different branching processes, but there is a unique, usually infinite, branching process obtained by unfolding "as much as possible". This branching process is called the unfolding of the P/T-net system. It is constructed to make verification of temporal properties possible.

In this section, we give a summary to basic notions and definitions related to net unfoldings we require for use in Section 6.5. Futher details can be found in Esparza and Heljanko (2008) (see also Engelfriet (1991) Esparza et al. (2002) and Khomenko et al. (2004)) which are concerned with the verification of properties.

Definition 6.1 (**Occurrence net**). *An occurrence net is an ordinary net* $ON = (B, E, G)$, *where* B *is a set of places of the net conventionally called conditions,* E *is a set of transitions conventionally called events and* G *is a flow relation.*

Two events of the unfolding of a net system are either connected by a path of net arcs or not. Thus:

Given two nodes (conditions or events), x *and* x' *of an* ON, *we say that:*

1. x *and* x', *are in structural conflict, denoted* $x\#x'$, *if there are distinct events* $e, e' \in E$ *such that* $\bullet e \cap \bullet e' \neq \phi$ *and* (e, x) *and* (e', x') *are in the reflexive transitive closure of the flow relation* G, *denoted by* $\preceq$.

2. A *node* x *is in structural self-conflict if* $x\#x$.

3. x *is causally related to* x' *denoted* $x \prec x'$, *if there is a (non-empty) path of arrows from* x *to* x' *(i.e.* x *must occur before* x'*).*

4. *two nodes are concurrent, denoted* $y \,co\, y'$, *if neither* $y\#y'$ *nor* $y \preceq y'$ *nor* $y' \preceq y$.

Figure 6.2(a, b) on page 92 shows occurrence nets where, e.g., the following relationships hold: $e_1 \prec e_6$, $e_7\#e_8$ *(due to the choice at* $b_5 \in \bullet e_4 \cap \bullet e_5$*) and* $e_6 \,co\, e_7$.

The occurrence net is characterised with the following:

- ON is acyclic (i.e., $\preceq$ is a partial order);

- for every $b \in B, |\bullet b| \leq 1$;

- for every $x \in B \cup E$, $\neg(x \# x)$ and there are finitely many x', such that $x' \prec x$, where $\prec$ denotes the irreflexive transitive closure of G

- $Min(ON)$ denotes the minimal w.r.t. $\preceq$ elements of $B \cup E$.

Definition 6.2 (**Homomorphism**). *A homomorphism from an occurrence net ON to a net system Σ is a mapping $h : B \cup E \longrightarrow P \cup T$ such that:*

- *$h(B) \subseteq P$ and $h(E) \subseteq T$ (conditions are mapped to places, and events to transitions);*

- *for all $e \in E$, the restriction of h to $\bullet e$ is a bijection between $\bullet e$ and $\bullet h(e)$ and the restriction of h to $e\bullet$ is a bijection between $e\bullet$ and $h(e)\bullet$ (transition environments are preserved);*

- *the restriction of h to $Min(ON)$ is a bijection between $Min(ON)$ and M_0 (minimal conditions correspond to the initial marking) and*

- *for all $e, f \in E$, if $\bullet e = \bullet f$ and $h(e) = h(f)$ then $e = f$ (there is no redundancy).*

In Figure 6.2(a, b) on page 92 presents two processes of the net system from Figure 6.1. In these processes, homomorphisms are indicated by labels inside nodes (for instance, $b_1, b_2 \ldots$ for conditions and $e_1, e_2, \ldots$ for events). .

Definition 6.3 (**Branching process**). *The set of branching processes of a net system Σ, is the smallest set of occurrence net ON satisfying the following conditions:*

1. *Let $M_0 = \{(p_1, \phi) \ldots, (p_n, \phi)\}$, where $\{p_1, \ldots, p_n\}$ is the set of initial makings of Σ. The occurrence net having M_0 as the set of condition and no events is a branching process of Σ.*

2. *Let π be a branching process of Σ such that some reachable marking of π enables a transition t. Let M be the set containing the places of the marking that are labelled by $\bullet t$. The occurrence net ON obtained by adding to π the event (t, M) and one condition $(p, \{(t,M)\})$ for every $p \in t\bullet$, is also a branching process of π . The event (t, M) is called a possible extension of π .*

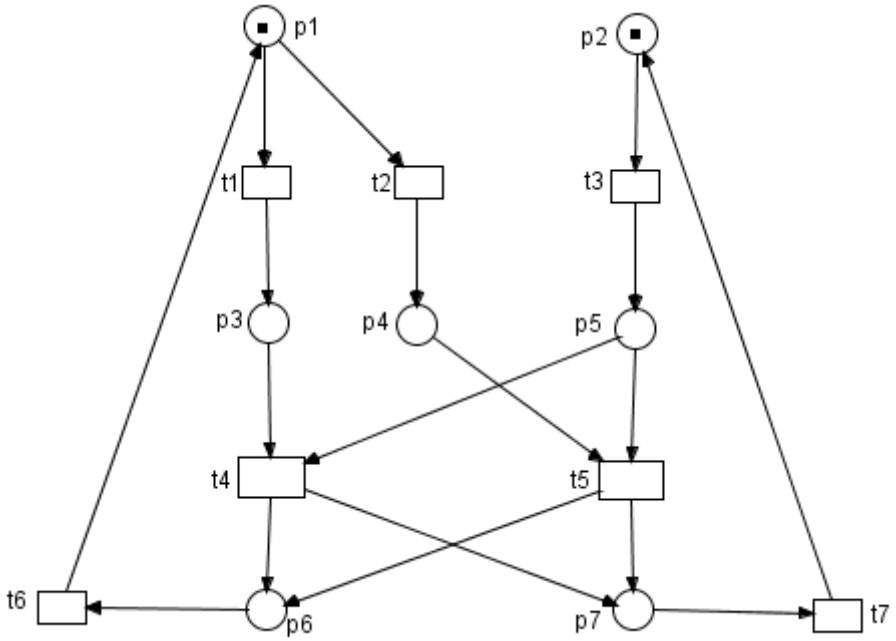

Figure 6.1.: A net system

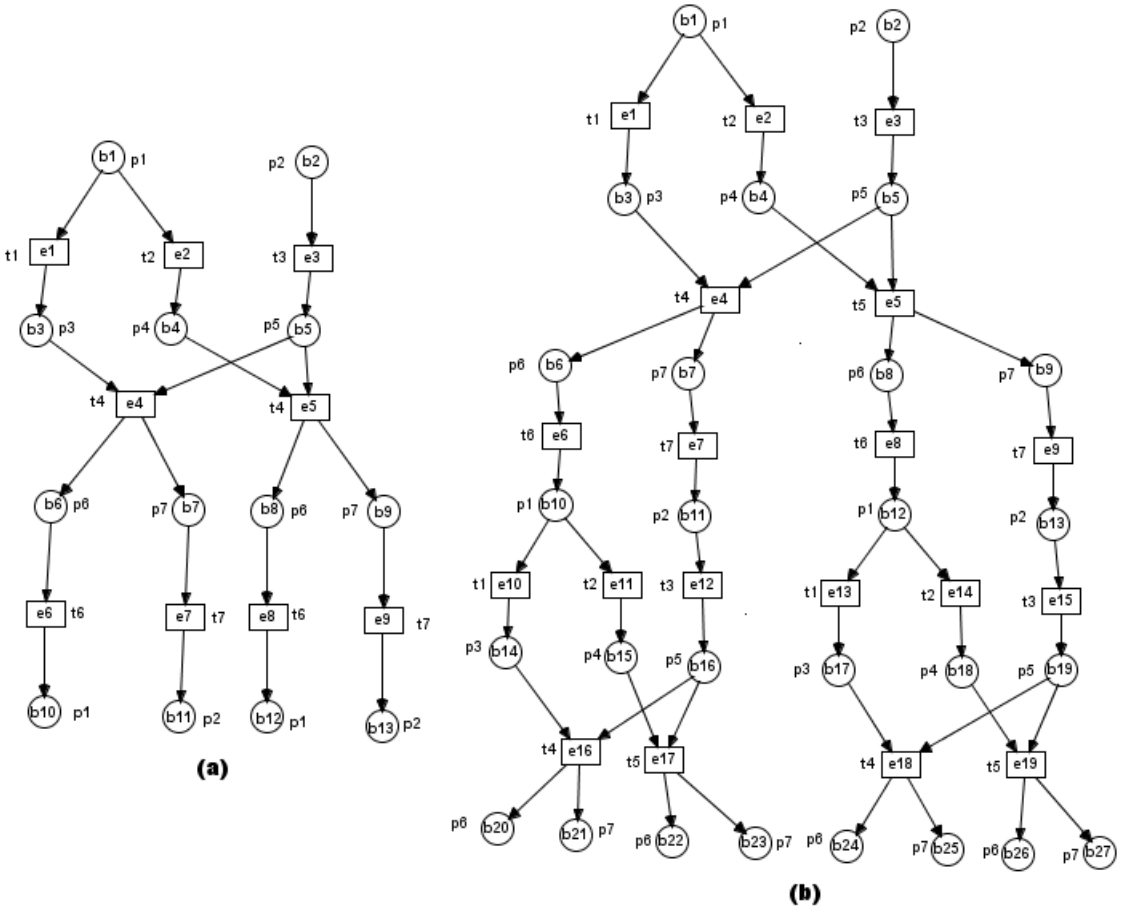

Figure 6.2.: Two branching processes for a system net of Figure 6.1

3. *If B is a (finite or infinite) set of branching processes of Σ, then so is the union $\bigcup B$.*

A branching process of a net system Σ is a quadruple $\pi = (B, E, G, h)$ such that (B, E, G) is an occurrence net and h is a homomorphism from it to Σ.

If an event e is such that $h(e) = t$ then e is said to be t-labelled. A branching process $\pi' = (B', E', G', h')$ of Σ is a prefix of a branching process $\pi = (B, E, G, h)$ of π, denoted $\pi' \subseteq \pi$, if (B', E', G') is a subnet of (B, E, G) containing all minimal elements and such that:

- *if $e \in E'$ and $(b, e) \in G$ or $(e, b) \in G$ then $b \in B'$;*
- *if $b \in B'$ and $(e, b) \in G$ then $e \in E'$; and*
- *h' is the restriction of h to $B' \cup E'$.*

The maximal branching process of a net system Σ w.r.t the prefix relation $\subseteq$ is called the *unfolding* of Σ and is denoted by Unf_{Σ}^{max}

Fundamental property of unfoldings
The fundamental property of Petri net unfoldings presented in Esparza and Heljanko (2008) states that: "the unfolding of the net system exhibits the same behaviour as the net system". Formally, it can be formulated as follows. Given two markings M_U, M'_U and an event e of the unfolding of Σ, the triple (M_U, e, M'_U) is called a *step* if M_U enables e and the occurrence of e leads from M_U to M'_U.

Definition 6.4 (**Fundamental property of unfoldings**). *Let M be reachable marking in a net system Σ, and let M_U be a reachable marking in Unf_{Σ}^{max} such that $h(M_U) = M$. Then*

- *If (M_U, e, M'_U) is a step of the unfolding, then there is a step (M, t, M') of Σ such that $h(e) = t$ and $h(M'_U) = M'$*
- *if (M, t, M') is a step of Σ, then there is a step (M_U, e, M'_U) of the unfolding such that $h(e) = t$ and $h(M'_U) = M'$.*

In other words, the fundamental property of unfoldings states that the reachable mark-

ings of the unfolding is isomorphic to the reachable markings of the original net system. Precisely, an unfolding of a given net system yields another net wherein places and transitions are labelled by the elements of the original net; therefore, a firing sequence or a reachable marking of an unfolding can always be interpreted in the context of the original net using the labels.

Configurations and Cuts

The notions of configurations and cuts have to be defined in the context of branching processes. This is due to the fact that a branching process can contain events in conflict. Therefore, a branching process is an acyclic net in which places must have at most one input transition but the number of output transitions is not constrained. Also, the environment of the transitions of the original net must be preserved. A configuration, is a set which is downward closed with respect to the causal relation of events and does not contain any pair of conflicting event. Formally it can be defined as follows.

Definition 6.5 (**Confuguration**). *A configuration of in a branching process π, is a set of events $C \subseteq E$ such that:*

- *for all $e, f \in C, \neg(e \# f)$ and,*
- *for every $e \in C$, if $f \prec e$ implies $f \in C$.*

For example, in the branching processes shown in Figure 6.2 (b)and (c) page 92 $\{e_1, e_3, e_4\}$ is a configuration whereas $\{e_1, e_2, e_3\}$ and $\{e_4, e_7\}$ are not ($\{e_1, e_2, e_3\}$ includes events in conflict, $e_1 \# e_2$, while $\{e_4, e_7\}$ does not include $e_1 \prec e_4$). Intuitively, a configuration is a partial-order execution, i.e., an execution where the order of firing of some of its events does not matter; e.g., the configuration $\{e_1, e_3, e_4\}$ corresponds to two totally ordered executions: $e_1 e_3 e_4$ and $e_3 e_1 e_4$.

The notion of cuts in a branching process is the counterpart of configurations as it focuses on conditions instead of events. In a branching process, any reachable marking is featured by a cut, which is a maximal (with respect to set inclusion) set of conditions that can be marked concurrently; in other words there is no causal relation between the set of conditions of a cut. Formally it can be defined as follows.

Definition 6.6 (**Cut**). *Let B' be a set of conditions and let C be a finite configuration of a branching process π. A cut is a maximal (w.r.t. $\subset$) such that:*

- *$b \, co \, b'$, for all distinct $b, b' \in B'$. and*

- *every marking reachable from $Min(\pi)$ is a cut.*

For instance, the set $\{b_6, b_7\}$ is a cut corresponding to the configuration $\{e_1, e_3, e_4\}$ of the branching processes of Figure 6.2 and the corresponding reachable marking of Σ is $\{p_6, p_7\}$.

As explained in the beginning of the section, the construction of an unfolding can be infinite, for every bounded net system Σ , if it is able to perform an infinite firing sequence. However, one can construct a finite complete prefix of the unfolding of Σ , by introducing an appropriate set E_{cut} of cut-off events to terminate the construction Such a complete prefix, in spite of being finite, contains sufficient information about the original net system. Thus there is no need to re-construct the full (potentially infinite) unfolding.

Definition 6.7 (**Completeness of branching processes**). *A branching process $\pi = (B, E, G, h)$ of Σ is said to be complete if there is a set $E_{cut} \subseteq E$ of cut-off events such that,*

- *for every reachable marking M of Σ, there exists a finite configuration C of π such that $C \cap E_{cut} = \phi$ and $M = Mark(C)$, and*

- *for each such C and every transition t enabled by M, there is an event $e \notin C$ in π such that $h(e) = t$ and $C \cup \{e\}$ is a configuration (e may be in E_{cut}).*

For example, the branching process shown in Figure 6.2(a) on page 92 is not complete since, e.g., the reachable marking $\{p_3, p_7\}$ is not represented in it. In contrast, the branching process in 6.2(b) is complete w.r.t. the set $E_{cut} = \{e_5, e_{16}, e_{17}\}$. Notice that the events e_8, e_9, $e_{13}e_{15}$, e_{18}, and e_{19} can be removed from the prefix without affecting its completeness. (This choice of E_{cut} is not unique: one could have chosen, e.g., $E_{cut} = \{e_4, e_{18}, e_{19}\}$'

Finally, it is worth noting that, we are not concerned with developing an algorithm for the construction of an unfolding. This is due to the fact that PUNF can generate the prefix of the unfolding. However, what we are interested in, is the acyclic nature of the unfolding. The acyclic nature permits the specification of procedure for verification of properties. Furthermore, unfolding approach alleviates the state space explosion problem more visual than state graphs and proven efficient for model checking.

Section 6.5 is devoted to how we specify a procedure for detecting the presence of a

deadlock strictly to safe nets into a Boolean satisfiability formulae by applying notions of configurations, cuts, and cut-off events on the complete prefix generated by PUNF.

6.3. Test cases

In this section, we describe some ordinary Petri net benchmark models which are used to test the performance of the developed tool for translating prefixes into propositional satisfiability formula and in the experiments performed in section 6.5. In most of our experiments the popular set of benchmark examples collected by Corbett (1996), K. McMillan, S. Melzer, S. Merkel, and S. Römer were used. Table 6.1 below, shows these models and their description. A more detailed description of these example models can be found in Corbett (1996) and (Melzer and Römer, 1997).

Table 6.1.: Concurrent Software Benchmark Models of Low-level Petri nets .

Model	**Model Name**	**Model Description**
BDS	Border Defense System	This example is the communication skeleton of a real Ada tasking program that simulates a border defense system. The example has 15 tasks, but the skeleton of each is relatively simple.
DME(n)	Distributed Mutual Exclusion	Distributed mutual exclusion asynchronous circuit with n cells.
DP((n)	Dining Philosophers	The standard version DP(n), which can deadlock.
DPD((n)	Dining Philosophers	the dictionary version, where the deadlock is prevented by having the philosophers pass a dictionary around the table.
DPH((n)	Dining Philosophers	The version with a host. There is an additional host task with which a philosopher must synchronize before attempting to acquire his forks.

Continued on next page

Table 6.1 – continued from previous page

Model column	Model Name column	Model Description column
ELEV(n)	Elevators	A model of a controller for a building with n elevators, using tasks to model the behaviour of the elevators themselves. The size n version has $n + 3$ tasks.
FTP(n)	File Transfer Program	A model of a program which services requests from n users to transfer files over a network. The size n version has $n + 8$ tasks.
FURN(n)	Remote Furnace Program.	This program manages temperature data collection for n furnaces. The size n version has $2n+6$ tasks.
GASQ(n)	Gas Station	This example models a self-service gas station. The model has one operator task, two pump tasks, and n customer tasks.
HART(n)	Hartstone Program	The communication skeleton of an Ada program in which one task starts and then stops n worker tasks.
KEY((n)	Keyboard Program	The communication skeleton of an Ada program that manages keyboard/screen interaction in a window manager. The program is scaled by making the number of customer tasks a parameter n. The size n version has n+5 tasks.
MMGT(n)	Distributed Memory Manager	he communication skeleton of an Ada program implementing the memory management scheme with n users. The size n version has n+4 tasks.
OVER(n)	Overtake Protocol	An Ada version of an automated highway system overtake protocol for n cars comprising $2n + 1$ tasks.

Continued on next page

Table 6.1 – continued from previous page

Model column	Model Name column	Model Description column
Q	User Interface	A model of an RPC client/server-based user interface with 18 tasks that is used by several real applications.
RING(n)	Mutual Exclusion Protocol	An Ada implementation of a standard distributed token ring mutual exclusion algorithm in which n user tasks synchronise access to a resource though m sever tasks that pass a token around a ring.
RW(n)	Readers and Writers	This example models a database that may be simultaneously accessed by any number of readers or a single writer. Each of the n reader tasks and n writer tasks must synchronise with a controller task before accessing and when fi nished accessing the database.
SENT(n)	Sensor Test Program	The communication skeleton of an Ada program that starts up n tasks to test sensors. The size n version has $n + 4$ tasks.
SPEED(n)	Speed Regulation Program	The communication skeleton of an Ada program that monitor and regulate the speed of a car.

6.4. Boolean Satisfiability

Propositional Satisfiability (SAT) is one of the classical problems in computer science. Its popularity started when it was proven to be an NP-complete problem in 1971 (Cook, 1971) and since that time, many algorithms were designed to solve this problem efficiently. Being an NP-complete problem, other well-known problems such as graph color-

ing, vertex cover, Hamiltonian path, and traveling salesman problem, of this class, can be encoded into a SAT instance giving them simpler representation. That makes SAT problem have a wide range of practical real-world applications. This include but not limited to Scheduling Crawford and Baker (1994), VLSI design Devadas (1989), testing problems Larrabee (1992), fault tolerance Barbour (1992), automated formal verification of hardware/software design Bryant et al. (1999) and a number of reasoning problems in artificial intelligence like planning Kautz et al. (1992).

The *Boolean Satisfiability problem* (SAT) consists in finding a satisfying assignment, i.e., a mapping $A : Var_\varphi \longrightarrow \{false, true\}$, defined on the set of variables Var_φ occurring in a given Boolean formula φ, such that φ evaluates to *true*. This formula is often assumed to be given in the conjunctive normal form $(CNF) \wedge_{i=1}^{n} = \vee_{l \in L_i} l$, i.e., it is represented as a conjunction of clauses, which are disjunctions of literals, each literal l being either a boolean variable or the negation of a boolean variable. It is assumed that no two literals in the same clause correspond to the same variable. The size of a formula in CNF is defined as the total number of literals in all its clauses, i.e., $\Sigma_{i=0}^{n} |l_i|$.

For example, the following corresponds to an instance of SAT:

$$\varphi = (x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3 \vee x_4 \vee \overline{x}_5) \wedge (x_1 \vee \overline{x}_3 \vee x_4).$$

A formula is said to be *satisfiable* if there is a truth assignment to its variables that makes it true. One possible assignment to the variables that will make the above formula satisfied is:

$$x_1 = true, x_2 = false, x_3 = true, x_4 = false, x_5 = false.$$

There are different types of SAT problems in the literature in terms of number of literals in a clause or occurrences of certain literals of variable. One famous type is the *k-SAT* where each clause has exactly k literals. It is already known that for $k = 2$, the formula can be solved in polynomial time (Papadimitriou, 1991) but for $k \geq 3$ the problem remains in NP-complete class. There is also the optimization version of this problem which is called *Max-SAT*. In this Max-SAT, the goal is to find the maximum number of clauses in the formula that can be satisfied. Due to the limited number of applications for this problem, it is not as famous as the decision one. Throughout the context of this chapter, the focus is on the decision version unless stated otherwise.

There are many algorithms developed for SAT problem and they have different kinds of strategies in order to reach to solution. For any SAT algorithm, there are two possible outcomes; either it gives an answer for both satisfiable and unsatisfiable instances with yes

or no answer respectively, or it can only give an answer for satisfiable instances. Therefore, SAT algorithms can be categorized into complete and incomplete algorithms. Some of the leading SAT solvers, e.g., MiniSAT Sorensson and Een (2005), can be used in the incremental mode, i.e., after solving a particular SAT instance the user can modify it (e.g., by adding and/or removing a small number of clauses) and execute the solver again. This is often much more efficient than solving these related instances as independent problems, because on the subsequent runs the MiniSAT can use some of the useful information collected so far(e.g., learnt clauses, see Wu and Fisher (1991)). Hence we chose to use the MiniSAT for our experiments.

6.5. Model checking using unfolding prefixes and SAT

One of the key analysis problems when analyzing concurrent and distributed systems that are model with Petri nets is that of deadlock-freedom: Do all reachable global markings of the net enable some transition? In this section we give an encoding of deadlock-freedom as a SAT problem and describe the translations of deadlock property into SAT formulae.

In model checking on Petri net unfoldings, a SAT instance φ of the net combined with the property to check is built using the prefix, such that:

- φ is unsatisfiable *iff* the property holds.
- every satisfiable assignment of φ gives a violating configuration
- φ has the form $CONF \wedge VIOL$, where CONF and VIOL are SAT formulae.
- Some of the variable of φ are associated with events of the prefix

The principle of the translation is to construct a propositional formula of a configuration wherein there is an event in (direct or otherwise) conflict for any cut-off event in the complete finite prefixes generated by an unfolder such as PUNF. Hence, obviously some dead markings are reachable from the reachable marking corresponding to the cut of such a configuration.

The procedure is illustrated for the branching process of the Dining Philosophers system. Figure 6.3 on page 101 shows the net system in (a), and its unfolding in (b). Initially, the configuration is set to the minimal configuration of an event in conflict with a given

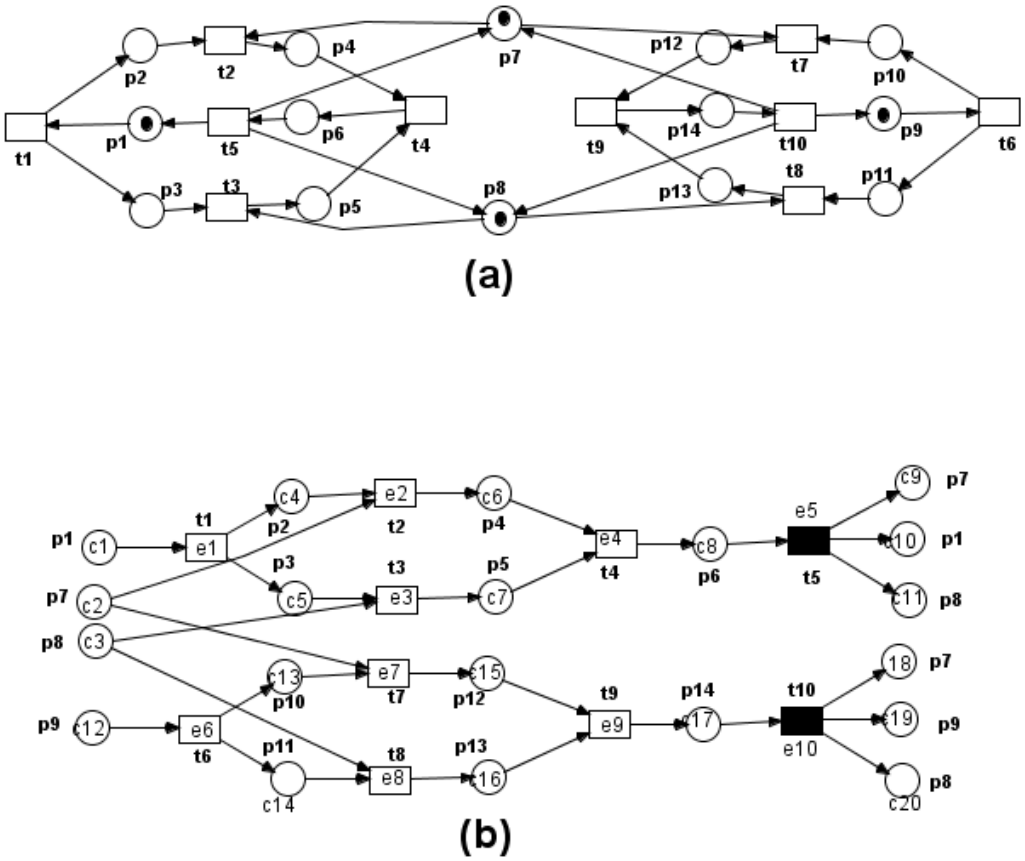

Figure 6.3.: (a) Dining Philosophers PN, (b) its unfolding

cut-off event. In our example, the configuration is set to the minimal configuration [e] of e_6 (in conflict with the cut-off e_5) as shown in Figure 6.4 (a) on page 102. To complete the procedure, an event in conflict with the cut-off e_{10} must be introduced into the configuration. Since e_1 satisfies these properties and because the union of its minimal configuration with the already formed configuration also yields to a configuration (there is no event in conflict), the procedure leads to the construction of the final marking of the configuration $\{e_1, e_6\}$. This is shown in Figure 6.4 (b) on page 102. From the corresponding reachable marking $p_1+p_7+p_8+p_9$ the dead marking $p_{10}+p_{11}$ is reachable.

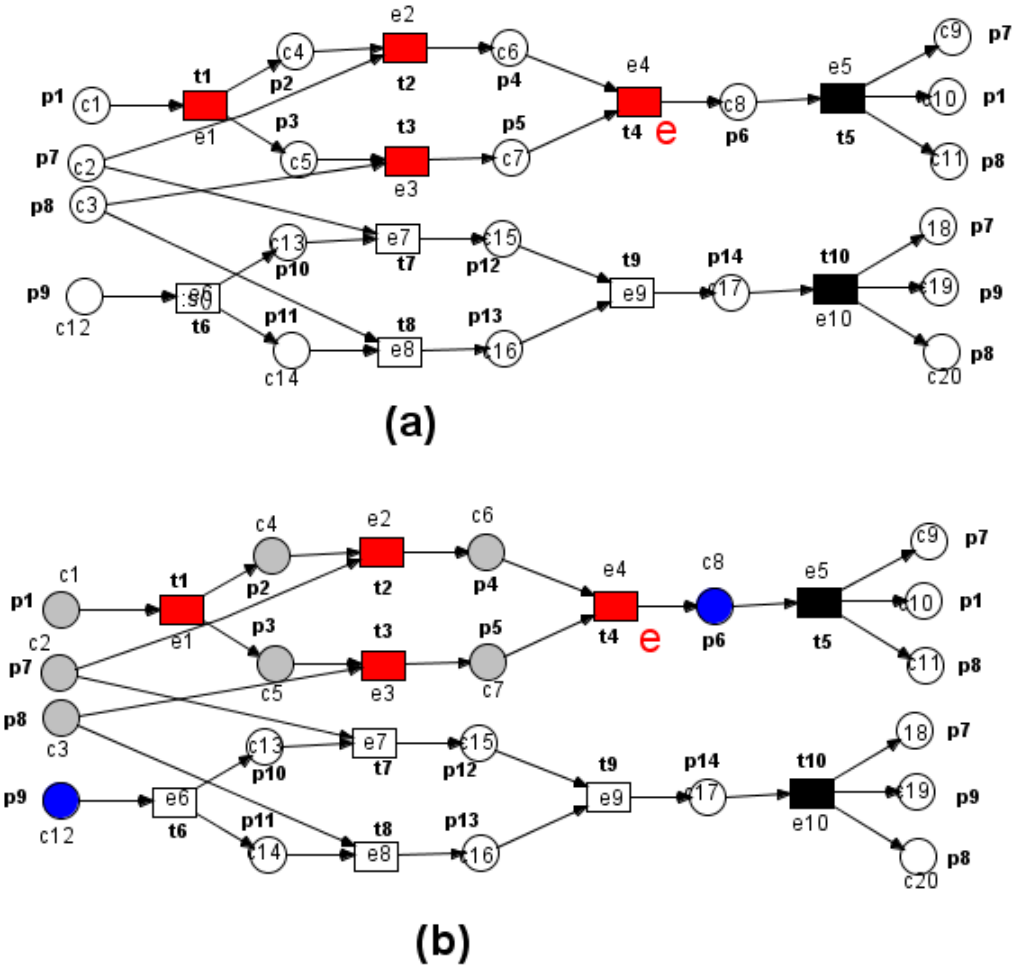

Figure 6.4.: (a) Minimal configuration, (b) Final marking of the configuration

Configuration constraint

At the level of a branching process, a deadlock configuration constraint is represented for each event $e \in E \setminus E_{cut}$, e and its immediate predecessors are in the configuration. Therefore the role of configuration constraints, which we represent as $CONF$, is to ensure that for each event in the set of all events in the prefix excluding cut-off events, its immediate predecessors are in the configuration, if the event is executed then all its direct causal predecessors are also executed, if the event is executed then no event in direct choice relationship with it can be executed. Also to ensure that it corresponds to the configurations C of the prefix (not just arbitrary sets of events). $CONF$ is formally defined as the conjunction of the formulae

$$CONF = \bigwedge_{e} \bigwedge_{f \in {}^{\bullet\bullet}e} (e \longrightarrow f) \wedge \bigwedge_{e} \bigwedge_{f \in ({}^{\bullet}e)^{\bullet} \setminus \{e\}} (\neg e \vee \neg f)$$

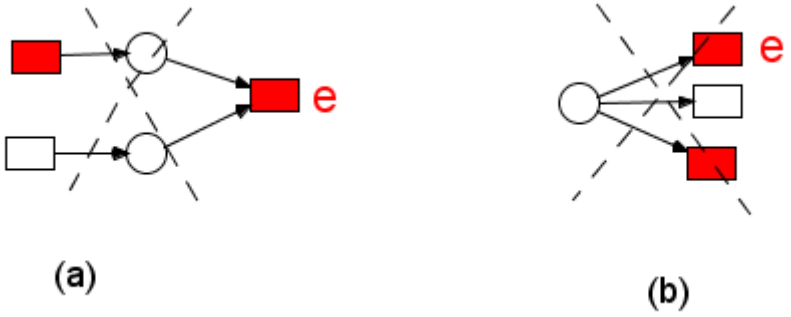

Figure 6.5.: Configuration Constraints

The above formula ensures that:

- if $e \in C$ then its immediate predecessors are also in C, i.e., C is downward closed w.r.t. $\prec$. Figure 6.5 (a) illustrates this notion.

- If e is executed then all its [direct] causal predecessors are also executed and if e is executed then no events in [direct] choice relationship with e can be executed (Figure 6.5(b)).

The satisfying assignments of $CONF$ correspond to the configurations of the prefix.

Deadlock violation constraint

The deadlock violation formula illustrated in Figure 6.6 on page 104 ensures that no event is enabled to fire, i.e. for every e: Figure 6.6 (a) presents this notion

- Some [direct] predecessor of e has not fired: see Figure 6.6 (b) or

- An event in [direct] conflict with e or e itself has fired: see Figure 6.6 (c).

The violation constraint is defined formally as follows:

$$VIOL = \bigwedge_{e} \Big(\bigvee_{f \in {}^{\bullet\bullet}e} \neg f \vee \bigvee_{f \in ({}^{\bullet}e)^{\bullet}} f \Big)$$

The method works for other reachability-like properties as well!

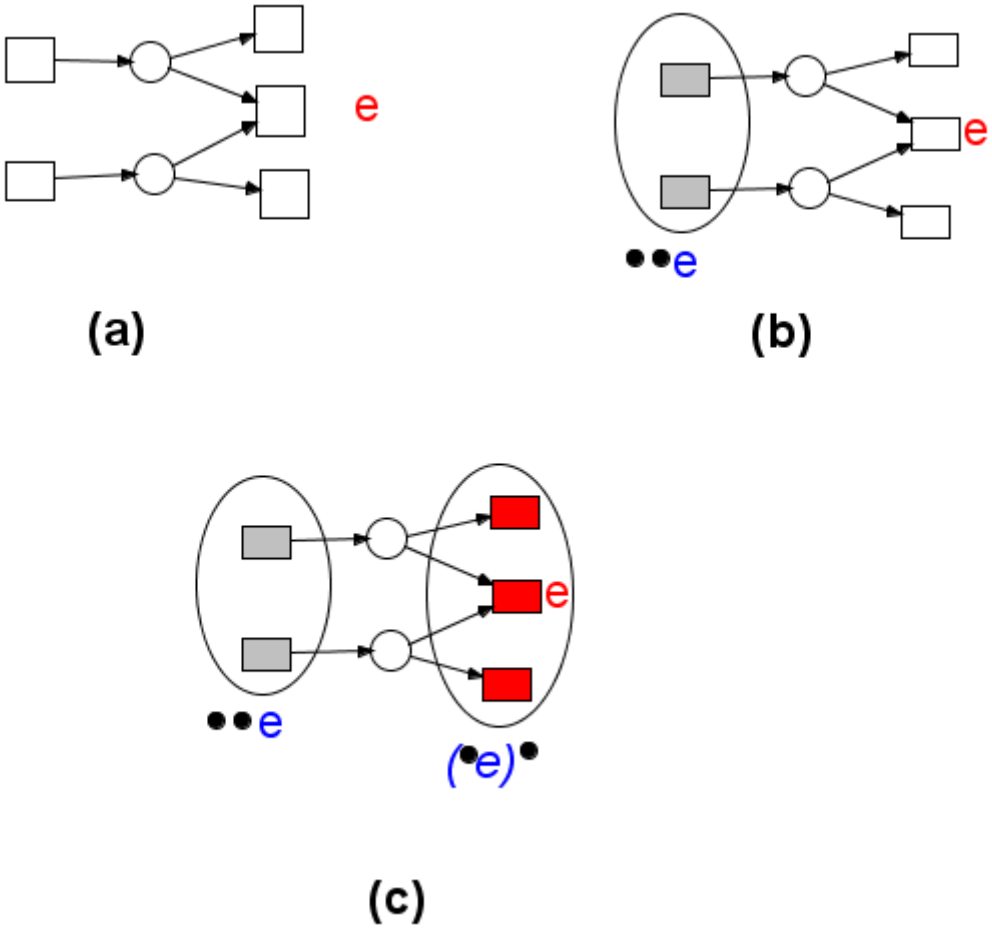

Figure 6.6.: Voilation Constraints

Translating to SAT

Finally, the problem at hand can be formulated as the SAT problem for deadlock:

$$Deadlock = CONF \wedge VIOL$$

6.6. Implementation

We have implemented both the deadlock and reachability checking translation to SAT in the tool PrefixtoCNF. We have ran experiments on three samples of low level Petri nets representing transformed ERS to test the tool. For deadlock checking only, we have made extensive benchmarking against other finite complete prefix based deadlock checking method to show the feasibility of our model checking method.

The PrefixtoCNF reads a file containing the description of a finite complete prefix generated by the PUNF - toolset. It then generates a CNF format file using the deadlock translation. After the translation has been created, the CNF file is then through an external interface given to the SAT solver to check whether a deadlock exists or not. If a deadlock exists, the SAT solver returns the answer *satisfiable* otherwise it returns the answer *unsatisfiable*. Depending on the option used when the SAT solver is invoked, the SAT solver can output a sequence of transitions (a set of assignments) which leads to a deadlock or to a counterexample marking.

In the followings, we describe the how the deadlock checking problem was translated into CNF file format, which is the input language for SAT solver in Java-like syntax. Thus the translation is relatively straight forward.

Listing 6.1, is a code snippet that shows how the configuration constraint respectively, the violation constraint were derived. Figure 6.7 on page 106 shows an example of a finite complete prefixes of a net system generated by PUNF which we give to the tool as input. Listing 6.2 is the code snippet that illustrates how the deadlock problem is encoded to CNF clauses, and Listing 6.3 shows the CNF generated for prefixes of unfolding of Figure 6.7.

MiniSAT, accepts its input in a simplified "DIMACS CNF" format which is a simple text format. Every line beginning "C" is a comment. The first non-comment line must be of the form:

p cnf NUMBER-OF-VARIABLES NUMBER-OF-CLAUSES

Each of the non-comment lines afterwards defines a clause. Each of these lines is a space-separated list of variables; a positive value means that corresponding variable (so 4 means $x4$), and a negative value means the negation of that variable (so -5 means $-x_5$). Each line must end in a space and the number 0.

So the CNF expression for our given problem is written as in Listing 6.3 without the comment line.

In Listing 6.3, the "p cnf" line means that this is a SAT problem in CNF format with 12 variables and 23 clauses. The first line after it is the first clause, meaning $-x4|x1$.

Someone can view this as a single expression. Alternatively, one can view this as a set of clauses, and the solver's job is to find the set of propositional variable assignments that

make all the clauses true.

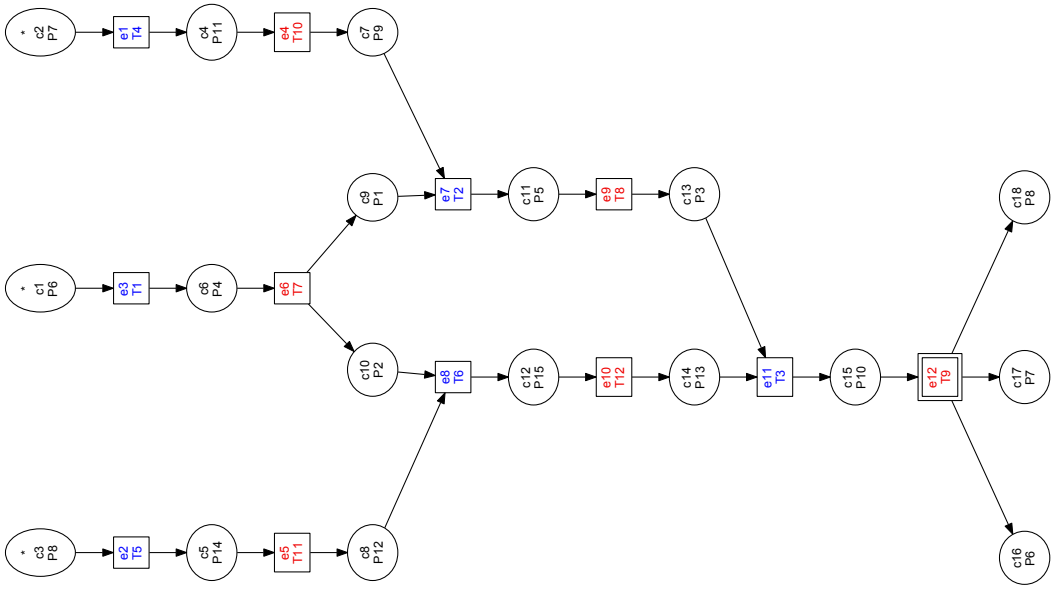

Figure 6.7.: An example complete finite prefixes of net system generated by PUNF

Listing 6.1: Create Configuration and Violation Constraints formulae

```
1
2    /* Read total number of conditions */
3    int number_cond = read4BytesNumber(aFile);
4
5    /* Read total number of events */
6    int number_event = read4BytesNumber(aFile);
7
8    /* create Event objects */
9    List<Event> events = new ArrayList<Event>();
10   for (int i = 0; i < number_event; i++)
11   {
12       /* Read original transition  numbers */
13       int transition_num = read4BytesNumber(aFile);
14       Event e = new Event(i + 1, transition_num);
15       events.add(e);
16   }
17   /* create condition objects */
18   List<Condition> conds = new ArrayList<Condition>();
19   for (int i = 0; i < number_cond; i++)
20   {
21    /* Read original place numbers *
```

```
22      int pl_num = read4BytesNumber(aFile);
23
24      /* read preset event number */
25      int preset_ev = read4BytesNumber(aFile);
26     Condition c = new Condition(i + 1, pl_num, preset_ev);
27     conds.add(c);
28     if (preset_ev != 0)
29        events.get(preset_ev - 1).postset.add(i + 1);
30        // here, read4BytesNumber(aFile) is the number
31        // of the  postset event of non-initial condition
32        for (int post = read4BytesNumber(aFile);
33                  post != 0; post = read4BytesNumber(aFile))
34        {
35           // add a condition postset (event number)
36           // to list of postset of conditions
37           c.postset.add(post);
38
39           // increament the location with the index
40           // (post-1)of preset list of an event that
41           //is the postset of the current condition
42           events.get(post - 1).preset.add(i + 1);
43        }
44
45     for (int buf = read4BytesNumber(aFile);
46                 buf != 0; buf = read4BytesNumber(aFile))
47     {
48        events.get(buf - 1).cutoff = true;
49
50        // read and ignore the corresponding event number
51        read4BytesNumber(aFile);
52     }
53
54     ArrayList<ArrayList<Integer>> clauses =
55                    new ArrayList<ArrayList<Integer>>();
56     for (Event e : events)
57     {
58        if (e.cutoff)
59        continue;
60        for (int c : e.preset)
61        {
62           if (conds.get(c - 1).preset_ev != 0)
63           {
64                 ArrayList<Integer> cl =
65                        new ArrayList<Integer>();
66              cl.add(-e.number);
67               cl.add(conds.get(c - 1).preset_ev);
68                 clauses.add(cl);
```

```
69              }
70          }
71      }
72
73      for (Event e : events)
74      {
75          if  e.cutoff)
76              continue;
77              for (int c : e.preset)
78              {
79                  for (int fnum : conds.get(c - 1).postset)
80                  {
81                        Event f = events.get(fnum - 1);
82                      if (f != e && !f.cutoff && e.number <f.number)
83                      {
84                            ArrayList<Integer> cl = new
85                          ArrayList<Integer>();
86                            cl.add(-e.number);
87                            cl.add(-f.number);
88                          clauses.add(cl);
89                          }
90                  }
91              }
92      }
```

Listing 6.2: Do Deadlock checking

```
1  /* DO: deadlocks */
2      for (Event e : events)
3  {
4          ArrayList<Integer> cl = new ArrayList<Integer>();
5          for (int c : e.preset) {
6          Condition cond = conds.get(c - 1);
7          if (cond.preset_ev != 0)
8              cl.add(-cond.preset_ev);
9              for (int fnum : cond.postset) {
10                 Event f = events.get(fnum - 1);
11                 if (!f.cutoff)
12                     cl.add(f.number);
13                 }
14             }
15         clauses.add(cl);// add new clause
16     }
```

Listing 6.3: Deadlock checking translation in DIMACS format for prefix in Figure-6.7

```
1      p cnf 12 23
```

```
2    −4  1  0
3    −5  2  0
4    −6  3  0
5    −7  4  0
6    −7  6  0
7    −8  5  0
8    −8  6  0
9    −9  7  0
10   −10  8  0
11   −11  9  0
12   −11  10  0
13   1  0
14   2  0
15   3  0
16   −1  4  0
17   −2  5  0
18   −3  6  0
19   −4  7  −6  7  0
20   −5  8  −6  8  0
21   −7  9  0
22   −8  10  0
23   −9  11  −10  11  0
24   −11  0
```

6.6.1. Experimental Results

We have carried out experiments using the bounded model checking approach with our tool by implementing it along with MiniSAT solver to test out the ideas set forth in this thesis. By doing so, we check for deadlock in each of the three samples. The results demonstrate how ERS can be model checked via transformation into P/T nets and subsequently, giving to an unfolder to generate the complete finite prefixes. Table 1 shows the running times for three different ERS that were transformed into 1-safe Petri nets by our tool (ERStoPTnet). Our experiments were conducted on a Pentium(R) 2.30GHz, 2.00GB RAM, 64-bit Operating System. Analysis times are reported in user CPU seconds collected using the *System.currentTimeInMillis()* command of Java. For the PrefixtoCNF results, the analysis times reported include the translation from the deadlock checking translation to SAT. For the MiniSAT tool, the analysis times include only the actual run times it takes the tool to check if deadlock exists or not. The rows of the table correspond to the three different models mentioned above. The columns represent: Problem, Net statistics, Prefix statistics, Formula statistics, Times in seconds and and whether a deadlock was found (DL). The other fields of the table are as follows:

$|P|$: number of places in the original net, $|T|$: number of transitions in the original net, $|B|$: number of conditions in the prefixes, $|E|$ numbers of events in the prefixes, $|E_{cut}|$: number of cut-off events, Var: number of variables in the SAT formula, Cl: number of clauses, Lit: number of literals, T_{pcnf}: time it takes PrefixtoCNF to translate the prefixes of unfoldings into SAT formulae, SAT: Time it takes the MiniSAT to check for deadlock on the input problem, DL: YES - the net system has a deadlock and NO means no deadlock.

Table 6.2.: Deadlock checking on transformed nets running times in seconds

Problem	**Net**		**Prefix**			**Formula**			**Time[s]**		**DL**
	$\|P\|$	$\|T\|$	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	Var	Cl	Lit	T_{pcnf}	SAT	
Samplenet1	22	20	34	17	0	21	55	148	0.004	0.01	YES
Lib-net2	12	7	14	6	0	9	19	48	0.005	0.01	NO
Lib-net4	48	36	136	68	3	39	220	520	0.006	0.03	NO

6.6.2. Tool evaluation

We have compared our experimental results using example benchmark models from Table 6.1, with another finite complete prefix based deadlock checking method. This method is the constraint-based logic programming by Heljanko (1999). It translates the problems of deadlock and reachability checking into the problem of finding a stable model of a logic program using finite complete prefixes. The implementation combines the prefixes of unfoldings, the translations, and an implementation of a constraint-based logic programming framework, in a tool called mcsmodels-toolset.

The mcsmodels reads a file containing the description of a finite complete prefix generated by the ERVunfold algorithm proposed in Esparza and Römer (1999), which is supported by PEP-toolset of Grahlmann (1997). It then generates a logic program using the deadlock or reachability translation, which is then, through an internal interface, given to the smodels stable model generator. After the translation has been created, the smodels computational engine is used to check whether a stable model of the program exists. If one exists, the program outputs a sequence of transitions which leads to a deadlock or to a counterexample state, using the found stable model. The experiments show that the constraint-based approach is quite competitive in terms of speed and space when

compared to previous prefix based deadlock checking algorithms such as the branch-and-bound deadlock detection algorithm by McMillan (McMillan and Probst, 1995), and the mixed integer programming approach by Melzer and Römer (Melzer and Römer, 1997). Thus we have chosen to use this method for comparative analysis to show accuracy of the functionality of our tool and to support the feasibility of our approach.

The performance of the comparative analysis technique depends on different factors, including the examples of the benchmark models to which they are applied, the time it takes each tool (mcsmodel or PrefixtoCNF) to perform the translation, and the property verified. The method for comparison depends on these factors in such away that the resulting performance data collected becomes meaningful by showing how efficient the tool PrefixtoCNF performs with respect to accuracy and speed.

Table 6.3 presents the example test case prefixes genereted by EVRunfolder and PUNF tools respectively with their corresponding average time for unfolding (creating a finite complete prefix). The time for EVRunfolder was measured using a Pentium II 267MHz, 512MB RAM, 128MB in the deadlock and reachability checking experiments conducted in Heljanko (1999). The time for PUNF was measured using Pentium(R) 2.30GHz, 2.00GB RAM, 64-bit Operating System. We do not have access to the PEP-toolset used in Heljanko (1999), our experiments with the PUNF toolset unfolding implementation seem to indicate that the computer we made our experiments on is approximately three times faster than theirs. This makes it difficult to comment on the absolute running times

The row of the table corresponds to all the example benchmark models applied. The columns represent: unfolding statistics for ERVunfolder, and PUNF. The other fields of the table are as follows: $|B|$: number of conditions in the prefixes, $|E|$ numbers of events in the prefixes, $|E_{cut}|$: number of cut-off events, unf: CPU time it takes to create a finite complete prefixes of particular model.

Table 6.3.: Benchmark model prefixes

Problem(size)	**Prefix by EVRunfolder**				**Prefix by PUNF**			
	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	unf	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	unf
BDS(1)	12310	6330	3701	2.4	1862	918	258	0.53
DME(4)	2381	652	16	0.5	2381	652	16	0.62
DME(5)	4096	1145	25	1.5	4096	1145	25	0.92
DME(6)	6451	1830	36	4.2	6451	1830	36	0.95
DPD(5)	6451	1830	36	4.2	6451	1830	36	0.37
DPD(6)	3786	1892	499	0.5	3786	1892	499	0.58
Continued on next page								

Table 6.3 – continued from previous page

Problem(size)	Prefix by EVRunfolder				Prefix by PUNF			
	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	unf	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	unf
DPD(7)	8630	4314	1129	2.2	8630	4314	1129	0.55
DPH(5)	2712	1351	547	0.2	2712	1351	547	0.48
DPH(6)	14590	7289	3407	4.1	14590	7289	3407	1.29
DPH(7)	74558	37272	19207	101.7	74558	37272	19207	4.29
ELEVATOR(2)	1562	827	331	0.1	4176	827	331	0.89
ELEVATOR(3)	7398	3895	1629	1.3	40766	3895	1629	2.44
ELEVATOR(4)	32354	16935	7337	27.4	373618	6935	7337	116.52
FTP(1)	178085	89046	35197	950.9	101871	50568	12643	7.36
FURNACE(1)	535	326	189	0.0	391	197	107	0.22
FURNACE(2)	4573	2767	1750	0.4	3033	1483	901	0.54
FURNACE(3)	30820	18563	12207	14.3	21396	10185	6575	1.72
GASQ(3)	2593	1297	490	0.3	2593	1297	490	0.45
GASQ(4)	19864	9933	4060	14.1	19864	9933	4060	1.31
HART(50)	354	202	1	0.1	354	202	1	0.88
HART(75)	529	302	1	0.1	529	302	1	2.16
HART(100)	704	402	1	0.2	704	402	1	2.63
KEY(2)	1310	653	199	0.1	623	294	30	0.87
KEY(3)	13941	6968	2911	4.8	4526	2237	322	1.64
KEY(4)	135914	67954	32049	398.3	43553	21742	4189	5.80
MMGT(3)	11575	5841	2529	3.3	11575	5841	2529	3.3
MMGT(4)	92940	46902	20957	308.5	92940	46902	20957	38.23
OVER(4)	1536	783	237	0.1	1536	783	237	0.38
OVER(5)	7266	3697	1232	1.8	7266	3697	1232	0.66
Q(1)	16123	8417	1188	14.8	16123	8417	1188	1.00
RING(7)	813	403	79	0.1	813	403	79	0.4
RING(9)	1599	795	137	0.2	1599	795	137	0.43
RW(9)	9272	4627	4106	0.5	9272	4627	4106	0.19
RW(12)	98378	49177	45069	25.3	98378	49177	45069	0.7
SENT(75)	533	266	40	0.1	538	266	40	0.58
SENT(100)	608	291	40	0.1	618	291	40	0.75
SPEED(1)	4929	2882	1219	0.7	4929	2882	1219	0.25

Table 6.4 presents the running times in seconds for our approach and mcsmodels approach presented in Heljanko (1999). The times have been measured as described above. The rows of the table corresponds to different problems. The columns represent: problem statistics, and tool statistics. The other fields of the table are as follows: DL: YES - the net system has a deadlock and NO means no deadlock, DC_{smo}: time for msmodels, average of five runs (The DCsmo columns also includes the logic program translation time, which was always under 10 seconds for the examples as reported in Heljanko (1999), DC_{SAT}: time it takes for SAT solver to check if a problem deadlock or not, average of five runs.

Both approaches were able to produce answers for all the examples presented here. Our approach was much faster in most cases because of the time it takes the SAT solver to check if a problem deadlock or not. For example, some problems that are deadlock-free like BDS, DME, DPH(5), DPH(6), FURNACE(1), FURNACE(2), and GASQ(3), takes the SAT solver less that one second, while it takes mcsmodels more than one second. Also, for the once that deadlock: ELEVATOR(2, 3, 4), HART(50,75, 100), KEY(2, 3, 4), Q(1) RING(7), RING(9), and RW(12) our approach was much faster as well. An important observation is that our approach produces exact answer to whether a problem deadlock or not similar to the mcsmodels approach. This means that on this problem set, our experiment shows that the functionality of the tool PrefixtoCNF is accurate, robust and competitive.

Table 6.4.: Deadlock checking running times in seconds

Problem(size)	**mcsmodel**		**PrefixtoCNF**	
	DL	DC_{smo}	DL	DC_{SAT}
BDS(1)	NO	1.4	NO	0.01
DME(4)	NO	1.4	NO	0.03
DME(5)	NO	4.8	NO	0.07
DME(6)	NO	13.6	NO	0.18
DPD(5)	NO	0.3	NO	0.02
DPD(6)	NO	2.0	NO	0.13
DPD(7)	NO	11.4	NO	1.73
DPH(5)	NO	0.6	NO	0.05
DPH(6)	NO	13.8	NO	2.00
DPH(7)	NO	324.9	NO	416.07
ELEVATOR(2)	YES	0.2	YES	0.02
ELEVATOR(3)	YES	4.3	YES	1.18
Continued on next page				

Table 6.4 – continued from previous page

Problem(size)	**mcsmodel**		**PrefixtoCNF**	
	DL	DC_{smo}	DL	DC_{SAT}
ELEVATOR(4)	YES	85.1	YES	171.1
FTP(1)	NO	702.9	NO	2586.55
FURNACE(1)	NO	0.09	NO	0.01
FURNACE(2)	NO	0.2	NO	0.06
FURNACE(3)	NO	2.3	NO	1.04
GASQ(3)	NO	0.8	NO	0.08
GASQ(4)	NO	51.5	NO	46.1
HART(50)	YES	0.1	YES	0.00
HART(75)	YES	0.1	YES	0.01
HART(100)	YES	0.2	YES	0.02
KEY(2)	YES	0.2	YES	0.03
KEY(3)	YES	17.8	YES	7.77
KEY(4)	YES	1287.2	YES	51.5
MMGT(3)	YES	6.1	YES	58.87
MMGT(4)	YES	523.4	YES	3440.05
OVER(4)	NO	0.1	NO	0.02
OVER(5)	NO	0.3	NO	0.39
Q(1)	YES	7.7	YES	4.73
RING(7)	YES	0.1	YES	0.03
RING(9)	YES	0.0	YES	0.06
RW(9)	YES	0.1	YES	0.1
RW(12)	YES	2.0	YES	0.01
SENT(75)	YES	0.0	YES	0.00
SENT(100)	YES	0.0	YES	0.0
SPEED(1)	YES	5.1	YES	1.68

6.7. Chaper summary

In Chapter 5, we described the implementation of a tool we developed which can automatically transform an ERS into safe-P/T nets. One of the outputs of this tool is a net

represented in Format_N which can be used for verification. In this chapter, presented how we carried out verification of reachability and deadlock properties for any net representing a transformed ERS described in Format_N. However, similar to other Petri net models, a transformed ERS also suffers from state space explosion problem. Using finite complete prefixes of net unfoldings as a way of alleviating the state space explosion problem has been very successful. Thus, we used bounded model checking with SAT solver via prefixes of unfolding to perform the verification. To be able to generate prefixes of the unfoldings, we have used an existing unfolding tool-set (The PUNF tool).

Our main contributions in this chapter were: (I) We devised translations of the problems of deadlock and reachability checking using finite complete prefixes into propositional satisfiability formulae. The procedure for these translations was based on finding the configuration and the violation constraints on the prefixes such that for each event in the set of all events in the prefix excluding cut-off events and its immediate predecessors, must be in the configuration. If the event is executed then all its direct causal predecessors are also executed, and no event in direct choice relationship with it can be executed and no event is enabled to fire if some direct predecessors of the event has not fired or an event in direct conflict with it or the event itself has fired. (II) The development of a generic tool called PrefixtoCNF. This tool when implemented can parse a file containing the description of a finite complete prefix of safe-Petri net in particular of a transformed ERS, and any bounded Petri net model in general, and subsequently generates an encoding for the above mentioned translations into a CNF. (III) The implementation of the above mentioned translations in the PrefixtoCNF is presented, with experimental results showing the accuracy of the functionality of the tool and also supporting the feasibility of our approach.

The PrefixtoCNF reads a file containing the description of a finite complete prefix generated by the PUNF. It then generates a CNF format file using the deadlock translation. After the translation has been created, the CNF file is then through an external interface given to the SAT solver to check whether a deadlock exists or not. If a deadlock exists, the SAT solver returns the answer *satisfiable* otherwise it returns the answer *unsatisfiable*. The division between the translation and the SAT solver made it mandatory for us to develop the PrefixtoCNF tool. Although, the scope of the tool was limited to translations of deadlock and reachability checking, it has been designed in such a manner that it can be extended to include the detection of other properties e.g. liveness, persistence, etc.

Chapter 7.

Conclusion

This chapter first summarises the contributions of the present thesis and then outlines open questions and future works.

Object Petri nets apply the concept of nesting to Petri nets which allows the idea of nested structure, mobility of objects, and by design of the transition occurrence rule, permits interaction between these objects. Despite the fact that object Petri nets are Turing-complete in their general form, they are suitable to model applications in, for example, an agent context, however verifying properties of the resulting model automatically, is in its embryonic stage. This is a drawback to the applicability of this formalism in practice.

The need for the application of bounded model checking with satisfiability solving via partial order (unfolding) approach for dynamic analysis of object net systems arose in Heitmann (2013) when it was suggested as a future study, to determined what requirement could be imposed upon EOS such that reachability and liveness properties can still be decided and the results would be beneficial for both the modeller and the verifier. The model of EOS has been modified in this thesis, and it has been proven that the main characteristics of the basic elementary object net systems, like boundedness and the decidability of reachability. It has also been shown that it is possible to transform the model into a 1-safe P/T net with equivalent behaviour and that model checking is possible in polynomial space.

The main goal of this thesis has been the development of a technique to systematically define the foundations for object Petri net transformation into P/T net in such a way that employing only affordable resources can handle important questions regarding the verification of various properties using model checking. This goal has successfully been

reached in terms of the contributions summarised below:

7.1. Contribution

- Theoretical contributions: object Petri nets have been extended to a new class of object net system which we called *Elementary Reference-net System*. To this end we established a set of rules for transforming an ERS into a behaviourally equivalent 1-safe P/T net in such a way that established tools can handle important questions regarding the verification of behavioural properties. Additionally, complexity results have been presented for the transformation. Among such results are the established Lemmas and proof of a theorem which relates the state space of 1-safe P/T nets and state space of 1-safe ERS.

- Development of ERStoPTnet: This is a software tool developed in Java for the transformation of ERS into P/T nets. The tool, executes the transformation algorithm and generates two different output formats: one of the outputs is a P/T net in PNML format for export to RENEW, which can be used for simulation and the other output is a textual P/T net representation in Format_N which is file format for low-level nets (LL) used as input for the Petri net unfolding tool-set.

- We devised general translations of the problems of deadlock and reachability checking using finite complete prefixes into propositional satisfiability formulae.

- Development of a generic software tool called PrefixtoCNF. This can parse a file containing the description of a finite complete prefix of unfolding of a transformed ERS, or any bounded Petri net model. The PrefixtoCNF can read a file containing the description of a finite complete prefix generated by PUNF and then generates a CNF format file using translations for deadlock and reachability detection problems. After the translation has been created, the CNF file is then through an external interface given to the SAT solver to check whether a deadlock exists or not.

- Application examples: some Petri net benchmark models have been encoded, including ERS example that model a simple library scenario where agents can move from hall to reading room pick up some books and transport them to the book-store where they are kept on shelves (this scenario illustrates how this formalism can be modelled to capture the concepts of nesting, mobility and interaction between these

objects). Experimental results have been presented supporting the feasibility of this approach.

7.2. Future work

This thesis has shown that the transformation of Object nets into an equivalent 1-safe Petri net allows bounded model checking. Nonetheless, some questions need to be addressed before verification for Object Petri net systems reaches the maturity of traditional model checking. In particular, open questions not considered in this thesis include:

- How to further extend the Elementary Reference-net system model by adding mechanism to create new objects. This is a major change in the formalism. In fact, the expansion toward 1-safe nets will not necessarily be possible if creation of objects is possible, and the state space will not be bounded anymore. As a result, decidability issues may arise because reachability properties which are decidable with the current model may become undecidable. Hence, this topic is probably a thesis in its own right.

- The formalism of the basic object net systems can be enhanced to allow for deeper nesting of net tokens (e.g. depth of more than two levels) in such a manner that net tokens may also move in the vertical dimension. Another future line of research is how we can apply the set of transformation rules defined in this thesis on such class of object net systems restricted to depth of higher level, which also allows to verify properties of the model.

- Another future line of research is how we can technically integrate the functionality of ERStoPtnet into the PUNF tool-set so the when it generates the textual net representation of the 1-safe equivalent to a given ERS then it can internally invoke the functionality of the tool to perform unfolding on the 1-safe net. Having this integration will allow users to utilise ERStoPtnet in real time application context.

Appendix A.

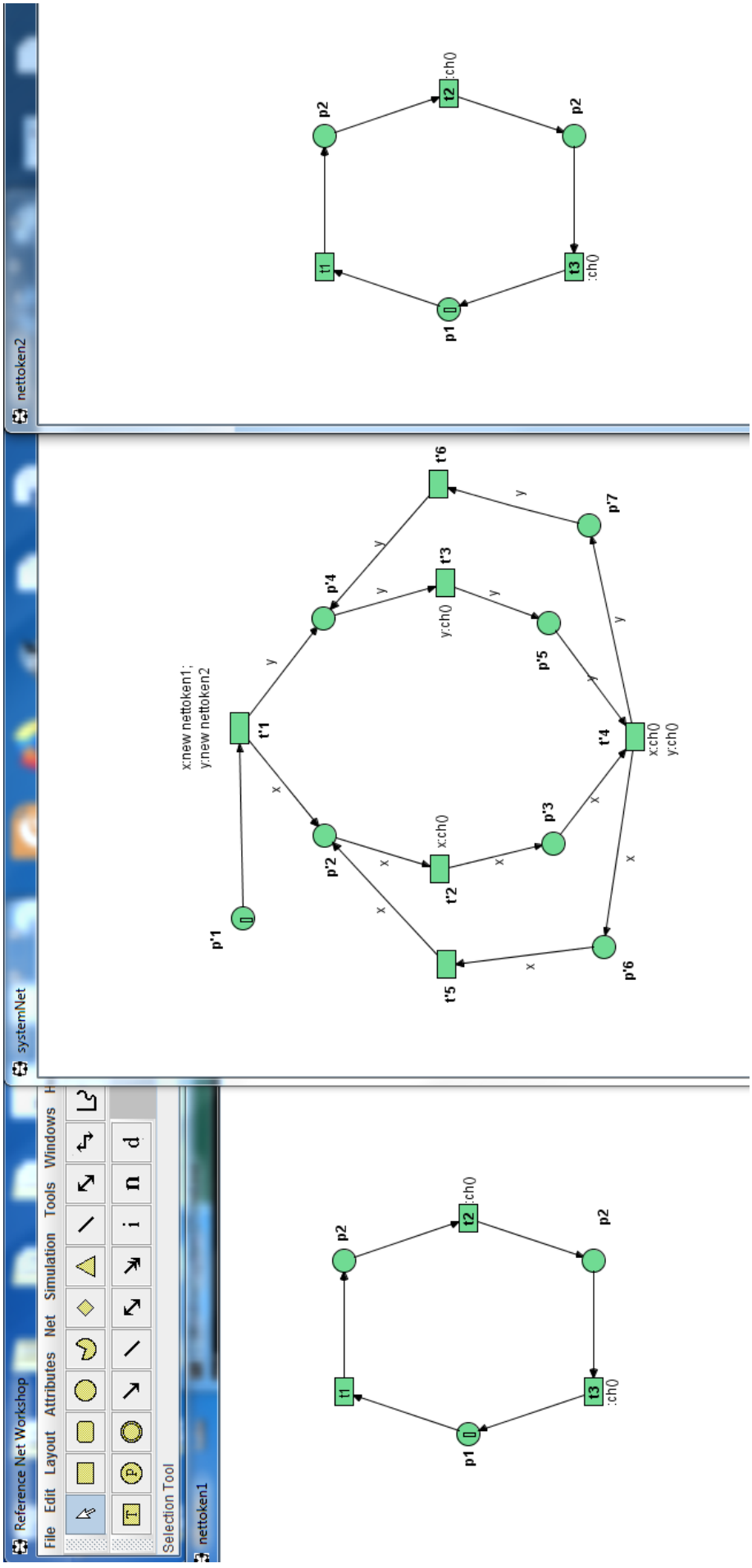

Figure A.1.: Example of ERS drawn in RENEW

Appendix B.

Listing B.1: PNML for example net in Appendix A

```
<pnml xmlns="RefNet">
<net id="netId1497633257780" type="RefNet">
   <place id="1">
      <name>
         <graphics>
            <offset x="-18" y="-24"/>
         </graphics>
         <text>p'1</text>
      </name>
      <initialMarking>
         <text>[]</text>
      </initialMarking>
   </place>
   <arc id="2" source="1" target="3">
      <type>
         <text>ordinary</text>
      </type>
      </arc>
   <transition id="3">
      <create>
         <text>x:new netToken1; y:new netToken2</text>
      </create>
      <name>
         <text>t'1</text>
      </name>
    </transition>
   <arc id="4" source="3" target="5">
      <inscription>
         <text>y</text>
      </inscription>
   </arc>
   <place id="5">
   </place>
   <arc id="7" source="3" target="8">
```

```
35          <inscription>
36            <text>x</text>
37          </inscription>
38        </arc>
39        <place id="8">
40            .
41            .
42            .
43        </place>
44        <arc id="9" source="5" target="10">
45        </arc>
46        <transition id="10">
47          <downlink>
48          <transition id="14">
49           <name>
50              <text>t'4</text>
51           </name>
52           <downlink>
53              <text>x:ch()</text>
54           </downlink>
55           <downlink>
56              <text>y:ch()</text>
57          </downlink>
58         </transition>
59          <name>
60          <text>systemNet</text>
61        </name>
62    </net>
63    <net id="netId1497633257860" type="RefNet">
64        <place id="1">
65            <name>
66              <text>p1</text>
67           </name>
68          <arc id="3" source="1" target="4">
69           <type>
70              <text>ordinary</text>
71           </type>
72        </arc>
73        <transition id="9">
74           <name>
75              <text>t2</text>
76           </name>
77           <uplink>
78               <text>:ch()</text>
79          </uplink>
80           .
81           .
```

```
82          .
83        <place id="11">
84        </place>
85        <arc id="12" source="11" target="13">
86        <transition id="13">
87           <name>
88              <text>t3</text>
89           </name>
90           <uplink>
91             <text>:ch()</text>
92           </uplink>
93       </transition>
94         </name>
95           <text>netToken1</text>
96        </name>
97     </net>
98     <net id="netId1497633257872" type="RefNet">
99        <place id="1">
100        <arc id="3" source="1" target="4">
101          .
102          .
103          .
104       <place id="11">
105       </place>
106       <arc id="12" source="11" target="13">
107       <transition id="13">
108       <text>netToken2</text>
109       </name>
110    </net>
111 </pnml>
```

Bibliography

Abdulla, P. A., Iyer, S. P., and Nylén, A. (2000). Unfoldings of unbounded petri nets. In *CAV*, volume 1855, pages 495–507. Springer.

Abdullahi, I. J. and Müller, B. (2016). Towards efficient verification of elementary object systems. In *CS&P 2016*, volume 247, pages 86–100. Humboldt University and CEUR.

Alsuwaiyel, M. H. (2016). *Algorithms: Design Techniques and Analysis (Revised Edition)*, volume 14. World Scientific.

Ansótegui, C., Bonet, M. L., and Levy, J. (2009). Solving (weighted) partial maxsat through satisfiability testing. *SAT*, 9:427–440.

Ansótegui, C., Li, C. M., Manyà, F., and Zhu, Z. (2012). A sat-based approach to minsat. In *CCIA*, pages 185–189.

Baldan, P., Corradini, A., König, B., and Schwoon, S. (2008). Mcmillans complete prefix for contextual nets. In *Transactions on Petri Nets and Other Models of Concurrency I*, pages 199–220. Springer.

Barbour, A. E. (1992). Solutions to the minimization problem of fault-tolerant logic circuits. *IEEE Transactions on Computers*, 41(4):429–443.

Billington, J., Christensen, S., Van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M. (2003). The petri net markup language: concepts, technology, and tools. In *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003. Proceedings*, pages 1023–1024. Springer.

Brassard, G. and Bratley, P. (1988). *Algorithmics: theory & practice*. Prentice-Hall, Inc.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691.

Bryant, R. E., German, S., and Velev, M. N. (1999). Microprocessor verification using efficient decision procedures for a logic of equality with uninterpreted functions. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 1–13. Springer.

Cabac, L. (2010). Modeling petri net-based multi-agent applications. `http://ediss.sub.uni-hamburg.de/volltexte/2013/6364/pdf/Dissertation.pdf`.

Cabac, L., Haustermann, M., and Mosteller, D. (2015). Renew-the reference net workshop. In *PNSE@ Petri Nets*, pages 313–314.

Cardelli, L. and Gordon, A. D. (1998). Mobile ambients. In *International Conference on Foundations of Software Science and Computation Structure*, pages 140–155. Springer.

Castagna, G., Vitek, J., and Nardelli, F. Z. (2005). The seal calculus. *Information and Computation*, 201(1):1–54.

Challenge, D. (1993). Satisfiability: Suggested format. *DIMACS Challenge. DIMACS*.

Clarke, E. M., Grumberg, O., and Long, D. E. (1994). Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542.

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM.

Corbett, J. C. (1996). Evaluating deadlock detection methods for concurrent software. *IEEE transactions on software engineering*, 22(3):161–180.

Crawford, J. M. and Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI*, volume 2, pages 1092–1097.

Devadas, S. (1989). Optimal layout via boolean satisfiability. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE MICROSYSTEMS TECHNOLOGY LABS.

Engelfriet, J. (1991). Branching processes of petri nets. *Acta Informatica*, 28(6):575–591.

Esparza, J. (1996). Decidability and complexity of petri net problemsan introduction. In *Advanced Course on Petri Nets*, pages 374–428. Springer.

Esparza, J. (1998). Reachability in live and safe free-choice petri nets is np-complete. *Theoretical Computer Science*, 198(1-2):211–224.

Esparza, J. and Heljanko, K. (2008). *Unfoldings: a partial-order approach to model checking*. Springer Science & Business Media.

Esparza, J. and Nielsen, M. (1994). Decidability issues for petri nets. *Petri nets newsletter*, 94:5–23.

Esparza, J. and Römer, S. (1999). An unfolding algorithm for synchronous products of transition systems. In *ConCur*, volume 1664, pages 2–20. Springer.

Esparza, J., Römer, S., and Vogler, W. (1996). An improvement of mcmillan's unfolding algorithm. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 87–106. Springer.

Esparza, J., Römer, S., and Vogler, W. (2002). An improvement of mcmillan's unfolding algorithm. *Formal Methods in System Design*, 20(3):285–310.

Esparza, J. and Silva, M. (1992). A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theoretical Computer Science*, 102(1):185–205.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. (2004). *Reasoning about knowledge*. MIT press.

Frumin, D. and Lomazova, I. (2014). Branching processes of conservative nested petri nets. In *Second International Workshop on Verification and Program Transformation*, pages 19–35. HSE Publishing House.

Girault, C. and Valk, R. (2013). *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer Science & Business Media.

Godefroid, P. and Wolper, P. (1991). Using partial orders for the efficient verification of deadlock freedom and safety properties. In *International Conference on Computer Aided Verification*, pages 332–342. Springer.

Goubault-Larrecq, J. and Mackie, I. (2001). *Proof theory and automated deduction*, volume 6. Springer Science & Business Media.

Grahlmann, B. (1997). The pep tool. In *Computer Aided Verification*, pages 440–443. Springer.

Hack, M. (1976). Petri net languages. `https://www.example.edu/paper.pdf`.

Heitmann, F. and Köhler-Bußmeier, M. (2012). A mobility logic for object net systems. In *Proceedings of the International Workshop on Logic, Agents, and Mobility (LAM12)*, volume 853, pages 19–34.

Heitmann, F. A. (2013). Algorithms and hardness results for object nets. `http://ediss.sub.uni-hamburg.de/volltexte/2013/6364/pdf/Dissertation.pdf`.

Heljanko, K. (1999). Deadlock and reachability checking with finite complete prefixes. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.8203`.

Holliday, M. A. and Vernon, M. K. (1987). A generalized timed petri net model for performance analysis. *IEEE Transactions on Software Engineering*, 12:1297–1310.

James, P. and Roggenbach, M. (2011). Automatically verifying railway interlockings using sat-based model checking. *Electronic Communications of the EASST*, 35.

Jensen, K. (1981). Coloured petri nets and the invariant-method. *Theoretical computer science*, 14(3):317–336.

Jensen, K. (2013). *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media.

Karp, R. M. and Rabin, M. O. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260.

Kautz, H. A., Selman, B., et al. (1992). Planning as satisfiability. In *ECAI*, volume 92, pages 359–363.

Khomenko, V. (2012). Punf documentation and user guide version 8.51 (parallel). `http://homepages.cs.ncl.ac.uk/victor.khomenko/home.formal/tools/punf/851/punf_manual_851.pdf`.

Khomenko, V. (2016). Punf-petri net unfolder. `http://homepages.cs.ncl.ac.uk/victor.khomenko/home.formal/tools/punf/851/punf_manual_851.pdf`.

Khomenko, V. and Koutny, M. (2001). Towards an efficient algorithm for unfolding petri nets. *CONCUR 2001Concurrency Theory*, pages 366–380.

Khomenko, V., Koutny, M., and Vogler, W. (2003). Canonical prefixes of petri net unfoldings. *Acta Informatica*, 40(2):95–118.

Khomenko, V., Koutny, M., and Yakovlev, A. (2004). Detecting state encoding conflicts in stg unfoldings using sat. *Fundamenta Informaticae*, 62(2):221–241.

Köhler, M. and Farwer, B. (2007). Object nets for mobility. *Petri Nets and Other Models of Concurrency–ICATPN 2007*, pages 244–262.

Köhler, M. and Rölke, H. (2004). Properties of object petri nets. In *ICATPN*, volume 3099, pages 278–297. Springer.

König, B. and Kozioura, V. (2008). Augur 2a new version of a tool for the analysis of graph transformation systems. *Electronic Notes in Theoretical Computer Science*, 211:201–210.

Kummer, O. (2002). *Referenznetze: Dissertation zur Erlangung des Doktorgrades am Fachbereich Informatik der Universität Hamburg*. Logos.

Kummer, O., Wienberg, F., Duvigneau, M., Schumacher, J., Köhler, M., Moldt, D., Rölke, H., and Valk, R. (2004). An extensible editor and simulation engine for petri nets: Renew. In *International Conference on Application and Theory of Petri Nets*, pages 484–493. Springer.

Larrabee, T. (1992). Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15.

Lewis, H. R. and Papadimitriou, C. H. (1978). The efficiency of algorithms. *Scientific American*, 238(1):96–109.

Lomazova, I. A. and Ermakova, V. (2016). Verification of nested petri nets using an unfolding approach. In *CEUR Workshop Proceedings*, pages 93–112. CEUR Workshop Proceedings.

Lomazova, I. A. and Schnoebelen, P. (1999). Some decidability results for nested petri nets. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 208–220. Springer.

Manber, U. (1989). Introduction to algorithms: A creative approach. addision-wesley. *Reading, MA*.

McMillan, K. L. (1992). Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *International Conference on Computer Aided Verification*, pages 164–177. Springer.

McMillan, K. L. and Probst, D. K. (1995). A technique of state space search based on unfolding. *Formal methods in system design*, 6(1):45–65.

Melzer, S. and Römer, S. (1997). Deadlock checking using net unfoldings. In *Computer Aided Verification*, pages 352–363. Springer.

Merlin, P. and Farber, D. (1976). Recoverability of communication protocols–implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043.

Milner, R. (1999). *Communicating and mobile systems: the pi calculus*. Cambridge university press.

Molloy, M. K. (1982). Performance analysis using stochastic petri nets. *IEEE Transactions on computers*, 31(9):913–917.

Nilsson, N. J. (1998). *Artificial intelligence: a new synthesis*. Elsevier.

Papadimitriou, C. H. (1991). On selecting a satisfying truth assignment. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 163–169. IEEE.

Petri, C. A. (1962). Kommunikation mit automaten. `http://edoc.sub.uni-hamburg.de/informatik/volltexte/2010/155/pdf/diss_petri_engl.pdf`.

Petri, C. A. (1966). Communication with automata. `http://edoc.sub.uni-hamburg.de/informatik/volltexte/2010/155/pdf/diss_petri_engl.pdf`.

Preiss, B. R. (1999). *Data structures and algorithms*. John Wiley & Sons, Inc.

Raimondi, F. and Lomuscio, A. (2004). Symbolic model checking of deontic interpreted systems via obdd's. In *LNCS*. Springer Verlag LNCS.

Rodríguez, C. (2013). *Verification based on unfoldings of Petri nets with read arcs*. PhD thesis, École normale supérieure de Cachan-ENS Cachan.

Rodríguez, C. and Schwoon, S. (2013). Cunf: A tool for unfolding and verifying petri nets with read arcs. In *International Symposium on Automated Technology for Verification and Analysis*, pages 492–495. Springer.

Rosa-Velardo, F. and De Frutos-Escrig, D. (2007). Name creation vs. replication in petri net systems. *Petri Nets and Other Models of Concurrency–ICATPN 2007*, pages 402–422.

Schwoon, S. and Römer, S. (2016). Mole - a petri net unfolder. `https://www.example.edu/paper.pdf`.

Sipser, M. (2006). *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston.

Sorensson, N. and Een, N. (2005). Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005(53):1–2.

Tarjan, R. E. (1987). Algorithm design. *Communications of the ACM*, 30(3):204–212.

Turing, A. M. (1938). On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, 2(1):544–546.

Valk, R. (1998). Petri nets as token objects. *Application and Theory of Petri Nets*, 1420:1–24.

Valk, R. (2003). Object petri nets: Using the nets-within-nets paradigm, advanced course on petri nets 2003 j. desel, w. reisig, g. rozenberg, eds., 3098. *Appendix A: Proof of Theorem*, 3.

Wagner, T. (2010). Prototypische realisierung einer integration von agenten und workflows. In *Informatiktage*, pages 101–104.

Weber, M., Kindler, E., et al. (2003). The petri net markup language. *Petri Net Technology for communication-based systems*, 2472:124–144.

Weiss, G. (1999a). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

Weiss, T. G. (1999b). Principles, politics, and humanitarian action. *Ethics & International Affairs*, 13:1–22.

Woolridge, M. (2001). Multi-agent systems: An introduction.

Wu, S.-F. and Fisher, P. D. (1991). Automating the design of asynchronous sequential logic circuits. *IEEE Journal of Solid-State Circuits*, 26(3):364–370.

Zhang, H. (1997). Sato: An efficient prepositional prover. *Automated DeductionCADE-14*, pages 272–275.